

戴冠平 编著

Tuxedo 企业级运维实战

清华大学出版社

叱咤风云：Tuxedo 企业级运维实战

戴冠平　编著

清华大学出版社
北　京

内 容 简 介

Tuxedo 是一个成熟多年的联机事务处理产品，用于开发、集成、部署和管理大型分布式应用。本书由浅入深地论述了 Tuxedo 的体系和理念，结合作者多年业内专家的从职经验，充分地剖析了 Tuxedo 的核心技术。对于 Tuxedo 在实际生产中，客户系统累积出现的各种典型故障和错误，分门别类地进行了透彻讲解，给出了具体的诊断思路和解决方案，具有非常现实、非常重要的指导意义和实战价值。

本书适合作为 Tuxedo 运维技术人员的参考手册，也可以作为高校相关专业师生学习资料。

图书在版编目（CIP）数据

叱咤风云：Tuxedo 企业级运维实战 / 戴冠平编著. —北京：清华大学出版社，2011.12
ISBN 978-7-302-26762-1

Ⅰ. ①叱…　Ⅱ. ①戴…　Ⅲ. ①互联网络–基本知识　Ⅳ. ①TP31

中国版本图书馆 CIP 数据核字（2011）第 185704 号

责任编辑： 夏兆彦
责任校对： 徐俊伟
责任印制：
出版发行： 清华大学出版社　　**地　　址：** 北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62795954，jsjjc@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 190×260　**印　　张：** 17.5　　**字　　数：** 437 千字
版　　次： 2011 年 12 月第 1 版　　**印　　次：** 2011 年 12 月第 1 次印刷
印　　数：
定　　价：　　元

产品编号：043914-01

前　　言

20 世纪 80 年代以来，伴随着网络和互联网的快速发展，中国各行各业的企业级信息系统应用从其体系结构、所采纳的技术和应用本身的复杂性都发生了深刻的变化。企业的 IT 系统日益成为其业务运营的支撑核心。主机、数据库、中间件、存储、网络等基础设施软硬件构成了企业 IT 系统的基石，同时这些不同层面的技术随着企业应用的规模扩张和复杂性增强，对企业的 IT 运维团队提出了越来越高的挑战，这也促使企业用户越来越深刻地认识到运维和服务所起的关键作用，及其带给自己的不菲价值。

在当今信息时代和人才时代，如何能够获得高质量的、快捷便利的 IT 服务是摆在所有企业 IT 运维团队面前的一个迫切需要思考的问题，而单纯求援于目前 IT 大厂商，其昂贵服务成本常常令大家觉得突兀和吃惊。一个可以作为参考的业内公开数据是，主流企业级软件公司的主要收入，尤其是利润来源，并不是软件销售本身，而是其技术和运维服务；而一般只够支付几年的服务运维费用，就已足够购买一套全新的软件产品。如果有个机会能够以相对公开的方式，针对企业级 IT 的运维进行系统性的知识整理和经验共享，相信这种努力和转移应当是件相当有益处并有意义的工作。

就我个人来讲，毕竟做了这么多年的中间件，很久很久以前，就一直想写这么一本书和大家分享；毕竟曾经写了那么多零零散散的心得，如浩瀚夜空中的点点繁星，错落杂乱；终于有了一大段空闲的时光，能让这些点点滴滴串通起来，努力组成一个有形状有系统的星座。

感谢当年引导和伴随我成长的 BEA 老同事们，让我有机会在中间件的领域里遨游，越游越深，虽然现在应该改称 Oracle 的各位新朋旧友了。也特别感谢联动北方的技术团队，正是你们的敬业精神，才激励我最终坚持把这本书编著出来；也正是你们的辛勤帮助，才有了这本书如此专业详实的内容。

独坐明窗映斜阳，听沧海潮起潮落，观碧空云卷云舒；恰逢母校百年风范依然，学子拳拳赤心依旧，“自强不息，厚德载物”。此书的及时完稿，也算是一丝慰籍和心意了。

作者

2011 年于 Brisbane Manly 海滨

目　　录

第 1 篇　入　门　篇

第 2 篇　基　础　篇

第 3 篇 实 施 篇

第 4 篇 诊 断 篇

第 5 篇 高 阶 篇

第 1 篇

入　门　篇

第 1 章　Tuxedo 概述

Tuxedo 是一个成熟多年的联机事务处理产品，用于开发、集成、部署和管理大型分布式应用，拥有处理关键业务应用系统问题所需的性能、安全性、可扩展性和高可用性，同时又易于安装、部署和管理。

1.1　什么是 Tuxedo 系统

Tuxedo（Transactions for UNIX，Extended for Distributed Operations）是在企业分布式计算环境中，开发和管理三层“客户机-服务器”（C/S）关键业务系统的平台软件。它具有空前的交易处理性能、高度的可靠性和无限的可伸缩性，能为企业建立、运行和管理大规模、高性能、分布式的关键业务系统提供强大的支撑平台。该平台具有很好的开放性，它支持各种各样的客户端、数据库、网络、通信方式和主机遗留系统。开发人员能够用它建立跨多个硬件平台、数据库和操作系统的应用系统。

图 1-1 展示了企业级 Tuxedo 系统的体系结构。

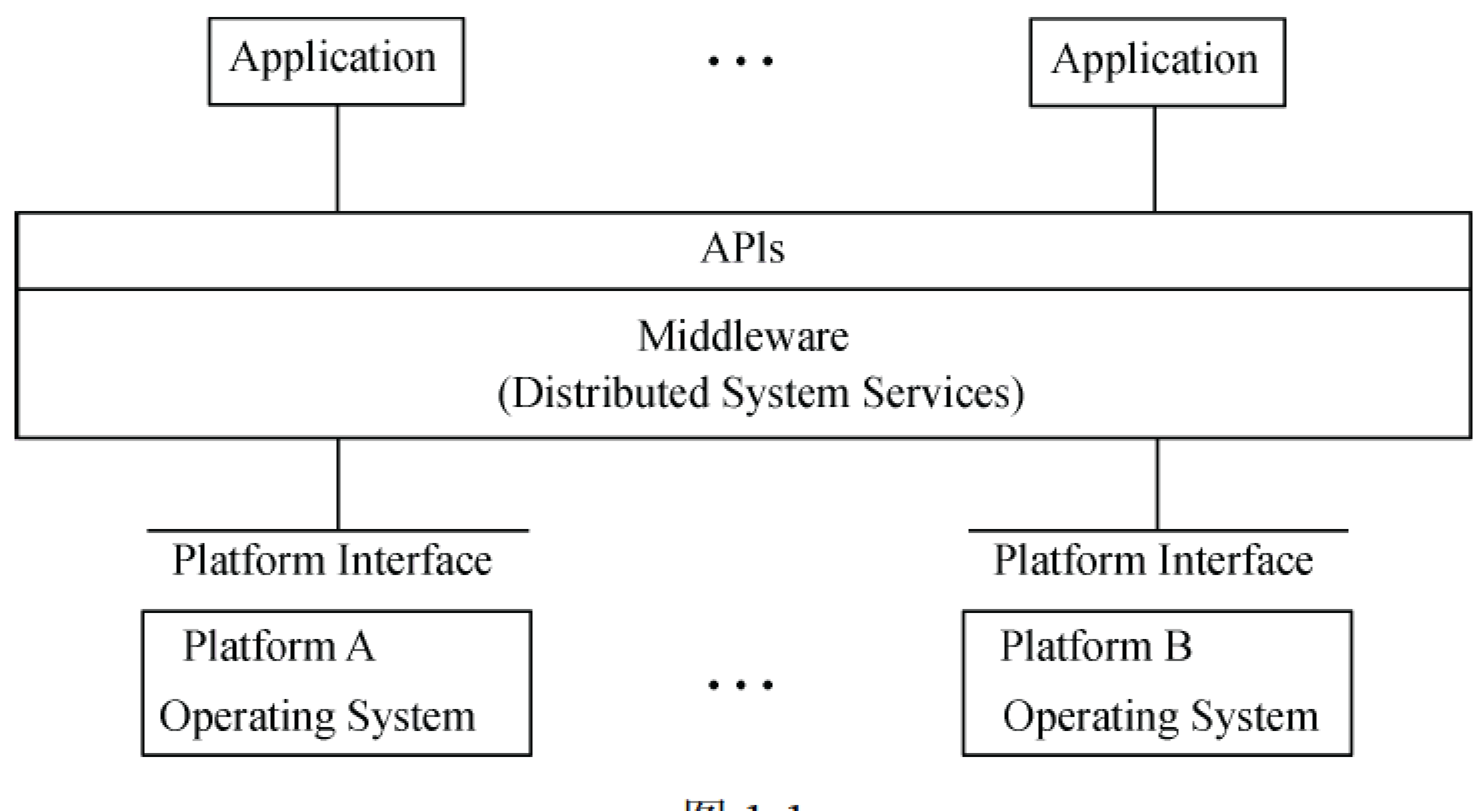


图 1-1

Tuxedo 具有全面而健壮的功能。在企业分布式联机交易系统中，Tuxedo 常作为一个事务监控器（TP Monitor，TM）来协调分布式事务；在构建多层 C/S 应用系统中，Tuxedo 常作为中间件的角色部署在客户机和服务器之间，为应用提供服务；在构建企业级应用系统中，Tuxedo 常以应用服务器平台的角色出现，为企业应用提供部署环境和运行环境。

Tuxedo 支持广泛的操作系统平台，包括 64 位/32 位的 Solaris，Linux，IBM 的 AIX、System i，HP 的 HP-UX、OpenVMS，以及 Microsoft 的 Windows。

1.2　Tuxedo 的历史及发展

Tuxedo 是一个久经考验的成熟的系统，在 20 多年的历史中不断地发展和增强。

1.1.1　Tuxedo 的产生

Tuxedo 系统于 1983 年由美国贝尔实验室的 AT&T 分部开发，最初被命名为 UNITS（UNIX Transaction System）。开发 UNITS 的目的是便于 AT&T 内部构建基于 UNIX 的业务支撑系统。

在 1989 年 UNITS 项目转移到 AT&T 的 UNIX 实验室（USL）时，这个 C/S 框架结构已经以“Tuxedo 系统”的名称销售了。1993 年 Tuxedo 系统被转到 Novell 公司。

在 1996 年，BEA 和 Novell 公司达成了排他协议来继续研发和出售不同平台下的 Tuxedo 系统，包括 Windows 和 UNIX 系统。2008 年 Oracle 公司收购了 BEA，Tuxedo 也转归 Oracle 旗下。

1.2.2　Tuxedo 的发展

1．从版本 1.0 到版本 7.1

从 1983 年的 1.0 版本到 2000 年的 7.1 版本，Tuxedo 系统经过无数次的改进和扩展，目的就是为了使客户端和服务器端的通信模式更加多样化。Tuxedo 系统作为事实上的标准，演变为开放式（open standard）的在线交易处理（OLTP）解决方案。Tuxedo 的 4.1 版本增加了 ATMI 接口以及对事务的支持。Tuxedo 对事务的支持直接导致了 XA 接口规范的产生。在 Tuxedo 的 5.1 版本中出现了域（Domain）组件，它能够实现 Tuxedo 系统中多个应用程序之间的动态链。Tuxedo7.1 版本推出了安全插件架构，这为集成第三方安全系统提供了接口。

2．版本 8.0

Tuxedo 系统的 8.0 版本发布于 2001 年，它的总体性能比其他版本有所加强，Tuxedo 8.0 的最大特点是引进了对 CORBA 的支持。在 CORBA 域中实现多线程、统一编程和负载均衡机制。Tuxedo 8.0 可以通过 WTC（WebLogic Tuxedo Connector）部件实现与 WebLogic 的互联。

3．版本 8.1

Tuxedo 8.1 版本发布于 2003 年，该版本对 WTC 做了进一步的加强，集成了 XML C++ 解析器，以便更好地支持 XML 数据。Tuxedo 可以和 WebLogic 7.1 或者更高版本的 Domain 集成，进行单点安全管理。还在本地化方面做了提高，除支持英语外，还支持了日语。网络通信方面，在没有改变任何接口的同时提高了域网关的性能。

4．版本 9.0

Tuxedo 的 9.0 版本发布于 2004 年，该版本主要是在 Web Service 方面做了进一步的加强，提供了 XML schema 和 FML 之间的双向转换功能，同时还提供了一个用于保存 Tuxedo

服务元数据的存储库（repository）。存储库的主要作用就是在应用开发阶段保存 Tuxedo 服务定义元素等信息，以方便开发人员进行交互式查询。Tuxedo 9.0 还在域网关的性能改进、超时控制、域连接策略、CORBA iiop client 故障转移等方面都做了很大的改进，在安全方面增加了 Cert-C PKI 插件以保护数据的安全，还增加了对 Kerberos 的支持，在 Microsoft .net 工作站客户机编程方面，为开发人员提供了一组 API 和开发工具。9.0 最大的特点是 JOLT 和 SNMP 代理开始和 Tuxedo 打包在一起销售，而不再是作为单独的产品部件。

5．版本 9.1

Tuxedo 9.1 版本发布于 2006 年，它作为在原来的版本基础上的一个升级版，增加了一些新的特性，在数据库方面，提供了对 Oracle RAC 的支持；在.NET Workstation 客户机方面，提供了一组实用的工具来帮助程序员快速地开发基于.net framework 的客户机应用程序；在管理方面，允许通过远程桌面启动、访问和关闭 Tuxedo 服务。

在 Tuxedo 9.1 版本发布的同时，还发布了 SALT（Service Architecture Leveraging Tuxedo），实现了 Tuxedo 应用与 SOA 的无缝集成。目前，SALT 已支持 Tuxedo 服务与 Web 服务的双向访问、数据转换、全局事务、安全控制、可靠消息等多种功能。同时，SALT 还支持 SCA（Service Component Architecture）服务组件架构，提供 SCA 容器，这样客户可以在强大的 Tuxedo 平台上构建或重用基于 SCA 规范的 SOA 应用。

6．版本 10.0

Tuxedo 的 10.0 版本发布于 2007 年，该系统最大的特点是增加了 TSAM（Tuxedo System and Application Monitor）应用管理监控平台。它由 Manager 和 Agent 构成，可以为 Tuxedo 应用程序提供全方位的性能监控和管理服务，根据事件产生告警，并进行性能调优，以满足高级服务需求。

Tuxedo 的安全方面在 10.0 版本中得到了很大的加强，首先是在安全验证方面，提供了通用的服务器 GAUTHSVR 来统一集成外部的 LADP 服务器。此前，Tuxedo 仅支持 LAUTHSVR 与 WebLogic 的内嵌 LDAP 服务器进行集成验证，而对于其他 LDAP 服务器则只能通过编码来实现。在认证加密方面，10.0 增加了 ATMI 应用对 SSL 的支持，而之前只有 CORBA 才支持 SSL。

在与第三方系统集成方面，10.0 版本提供了 MQ adapter 与 Websphere MQserices 进行双向的基于事务的连接和消息交换服务。Tuxedo 10.0 在其他方面的更新还包括提供 tpkill 命令来关闭未响应的进程，以免公告板共享内存区的数据结构遭到破坏，而不是像以前那样通过 kill 来发送 SIGKILL 信号。同时它还提供了对大文件的支持，所有的 UNIX 平台上均支持大于 2GB 的文件。

7．Tuxedo 10gR3

Tuxedo 10gR3 版本发布于 2009 年。它在网络通信协议方面增加了对 IPv6 的支持。IPv6 是由 IETF 设计的下一代协议，用来取代当前的 IPv4 网络协议。IPv6 中最明显的改进是 IP 地址长度从 IPv4 的 32 位加长到 128 位，它也改进了诸如路由和网络自动配置等许多方面。

Tuxedo 10gR3 版本增加了新的 API，支持在服务进程中创建用户 context，之前用户只

能在系统 context（Tuxedo 服务被调用时自动创建的 context）中发送/接收服务请求和定义全局事务。

为帮助 Tuxedo 管理员监视客户端的运行合法性，Tuxedo 从 10gR3 版本开始提供访问日志，可以记录访问的最高客户量、当前的用户访问量和指定的用户。

Tuxedo 10gR3 版本还进行了如下改进。

- **CLOPT 长度** 在 UBBCONFIG 中 Tuxedo ATMI SERVER 的 CLOPT 长度从原来的 256 增加到 1024。
- **FML/FML32 字段名长度** FML/FML32 字段名长度从 30 增加至 254。
- **tlisten 密码加密** tlisten 的密码经过系统加密后再保存到 tlisten.pw 文件，此前这是明文保存的。
- **动态 DMIB 更新** 允许重新配置远程域网关侦听地址，而无需重启本地域。
- **域网关永久不连接策略** 增加新的连接策略 PERSISTENT_DISCONNECT，表示既不主动连接其他域，也不接受其他域的连接。

8. Tuxedo 11g

Tuxedo 11g 发布于 2010 年。在该版本中新增的客户端-服务器端亲和性的功能，使得 Tuxedo 客户端可以指定某一段时间内所有交易请求的路由范围，可以是严格的范围，也可以是倾向的范围。

Tuxedo 11g 支持在一个服务器组中定义多个资源管理器（RM），这样每个应用服务器都可以在一个全局事务中使用多个资源管理器。

Tuxedo 11g 版本还支持在 Tuxedo VIEW 数据结构中再嵌套 VIEW。

先前的 Tuxedo 版本支持为指定服务设定 AUTOTRAN，使得该服务被调用时如果尚不在全局事务中，则自动开始全局事务。Tuxedo 11g 版本中增加了域级别 AUTOTRAN 的配置。

1.3 Tuxedo 支持的平台

Tuxedo 支持现在市场上主流的操作系统平台。

表 1-1 提供了 Oracle Tuxedo 11g Release 1（11.1.1.2.0）支持的平台详细信息。

表 1-1

生产商	操作系统	版本
HP	HP-UX	HP-UX 11i v2 (32-bit) on Itanium 64-bit
		HP-UX 11i v2 (64-bit) on Itanium
HP	HP OpenVMS	HP OpenVMS HP OpenVMS V8.3-1H1 (64-bit) on IA64
IBM	AIX	IBM AIX 5.3 (32-bit) on IBM PowerPC
		IBM AIX 5.3 (64-bit) on IBM PowerPC
		IBM AIX 6.1 (32-bit) on IBM PowerPC
		IBM AIX 6.1 (64-bit) on IBM PowerPC

续表

生产商	操作系统	版本
IBM	IBM i	IBM i 6.1 on IBM PowerPC
Microsoft	Windows	Microsoft Windows 2008 Server (32-bit) on x86 with MS Visual Studio 2008 Professional Edition Microsoft Windows 2008 Server (64-bit) on x86-64 with MS Visual Studio 2008 Professional Edition Microsoft Windows 7 on x86 with Visual Studio 2008 (Client Only) Microsoft Windows XP on x86 with Visual Studio 2008 (Client Only)
Novell	Linux	Novell SUSE Linux Enterprise Server 10 (32-bit) on x86 Novell SUSE Linux Enterprise Server 10 (64-bit) on x86-64 Novell SUSE Linux Enterprise Server 10 (64-bit) on s390x Novell SUSE Linux Enterprise Server 11 (32-bit) on x86
Oracle	Linux	Linux　Oracle Enterprise Linux 5.0 (32-bit) Oracle Enterprise Linux 5.0 (64-bit)
Red Hat	Linux	Red Hat Linux Enterprise AS 5 (32-bit) on x86 Red Hat Linux Enterprise AS 5 (64-bit) on s390x
Sun Microsystems	Solaris	Sun Microsystems Solaris 10 (32-bit) on SPARC Sun Microsystems Solaris 10 (64-bit) on x86-64 Sun Microsystems Solaris 10 (64-bit) on SPARC

1.4　Tuxedo 的技术架构

企业级的应用大致经历了 3 个阶段：以大型机为核心的“主机/终端”模式，以文件服务为核心的“文件服务器”模式和以数据服务为核心的“客户机/服务器”模式。

“主机/终端”模式属于单层结构，它的典型代表是 IBM OS/390 系统，所有数据、应用逻辑、用户界面都放在主机上，终端机不具有持久存储和计算能力。执行应用程序时，终端将用户界面下载到本地内存中，处理能力完全依赖主机的吞吐量和通信网络的带宽。

“文件服务器”模式属于两层结构，它的主要代表是 Novell Netware 网络操作系统，所有文件和数据都存储在服务器上，客户机是功能完善的计算机系统，负责提供用户界面并处理业务逻辑。

“客户机/服务器”模式可以分为以数据库管理系统（DBMS）为核心的两层结构和以中间件（application server）为核心的多层结构。基于 Tuxedo 中间件构建的应用系统就具有多层可管理的客户机/服务器框架结构。

1.4.1　客户机/服务器模式

在客户机/服务器模式下，企业应用被分割成若干个相对独立的功能模块，其中一部分模块用来管理企业的系统资源并为其他模块提供服务，称为“服务器”；另一部分模块为用

户提供输入输出界面，向服务器提交请求，称为“客户机”。客户机/服务器模式如图 1-2 所示。

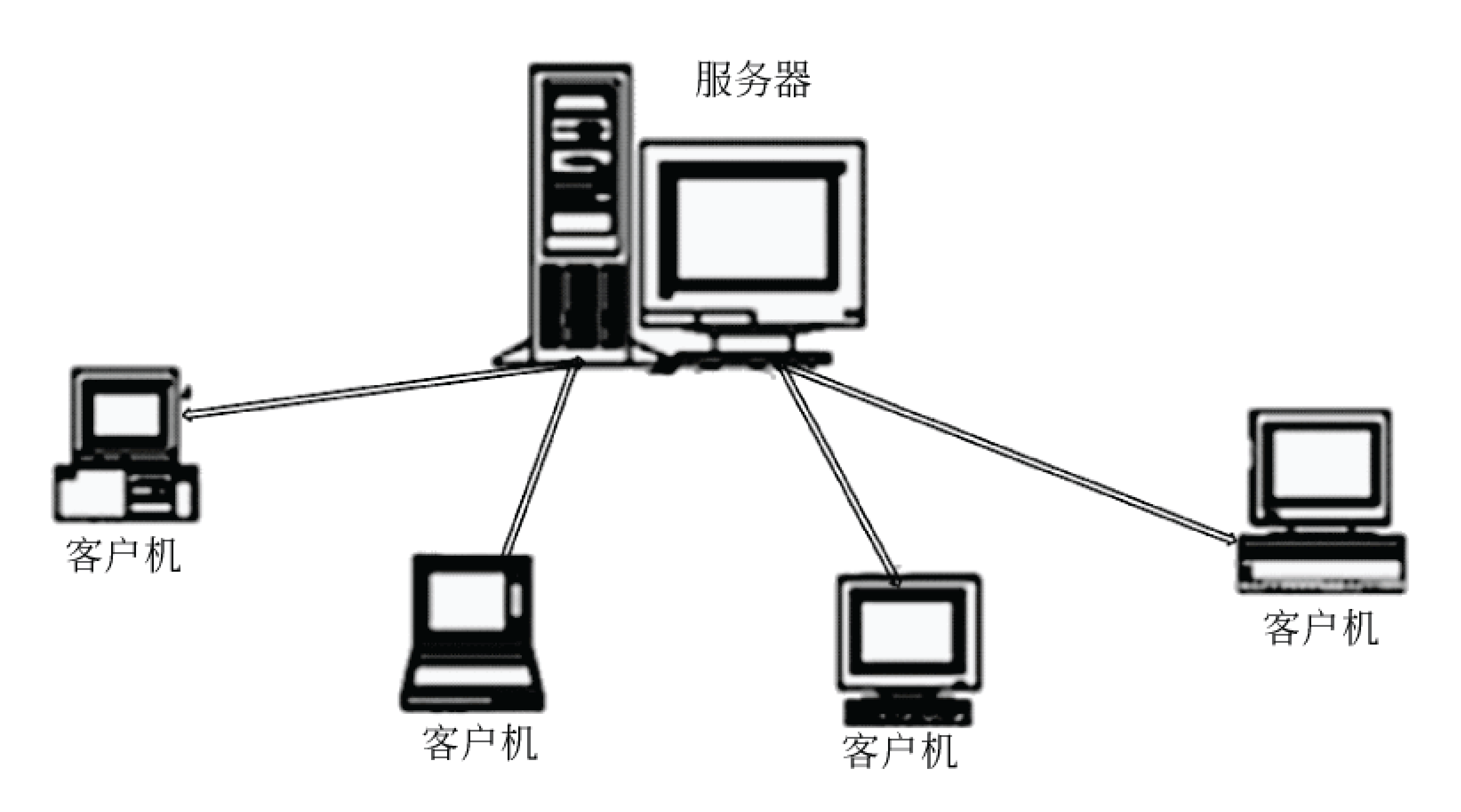


图 1-2

随着企业计算的发展，客户机/服务器模型经历了一个从以 DBMS 为核心的两层结构向以应用服务器为核心的多层结构演变的过程。

1．两层客户机/服务器模型

典型的两层客户机/服务器模型为：服务器端运行关系型数据库，负责提供数据服务，所有应用逻辑和用户界面都安装在客户端，客户机与服务器通过“请求/应答”式通信模型（Request/Response）进行业务逻辑处理，大大减少了网络流量。与单层结构相比，两层客户机/服务器模型提高了执行效率，减少了网络通信流量，加强了模块化设计和分布式计算能力。但是随着应用规模的扩大，它的缺点也就逐渐暴露出来。

- **两层客户机/服务器结构无法处理大并发的用户请求**　服务器没有提供连接池管理机制，连接不能在多个客户端共享，也就是每个客户端都要和数据库建立连接。这就消耗了大量的系统资源，当并发的用户数增加到一定的数量的时候，就可能造成死锁或者系统崩溃。
- **系统的可移植性差**　编程环境往往依赖于操作系统和数据库，客户机编程语言和数据库 SQL 对平台和产品的依赖性都很强，在操作系统或数据库发生改变的情况下，往往需要重写整个系统。
- **系统的灵活性差**　虽然系统分成了客户机/服务器两个部分，但它们是紧耦合的，当其中一方发生改变时，通常会影响到另一方。
- **不支持分布式事务**　两层客户机/服务器结构没有分布式事务的管理能力，当一笔交易涉及到多个资源管理器时，不能保证交易的一致性。
- **安全性差**　存放应用程序的客户机容易受到黑客的攻击，每个客户机都可以直接访问数据资源，这样会对企业级资源安全构成威胁。
- **可维护性差**　每个客户机上都要安装操作系统和应用软件，这不但造成了巨大的资源浪费，而且系统升级时工作量会非常大。

- **可管理性差** 对于客户机而言，不支持集中式事务管理；对于服务器而言，没有统一的管理工具，这样就不便于管理。
- **可扩展性差** 两层结构的可扩展性非常有限，这类系统的容量一般由设计时决定，项目实施后扩展很慢。

2. 三层客户机/服务器模型

把在两层结构中部署的客户机和服务器上的业务逻辑抽取出来，单独放在一个中间层上去处理，这样就构成了三层结构，如图 1-3 所示。

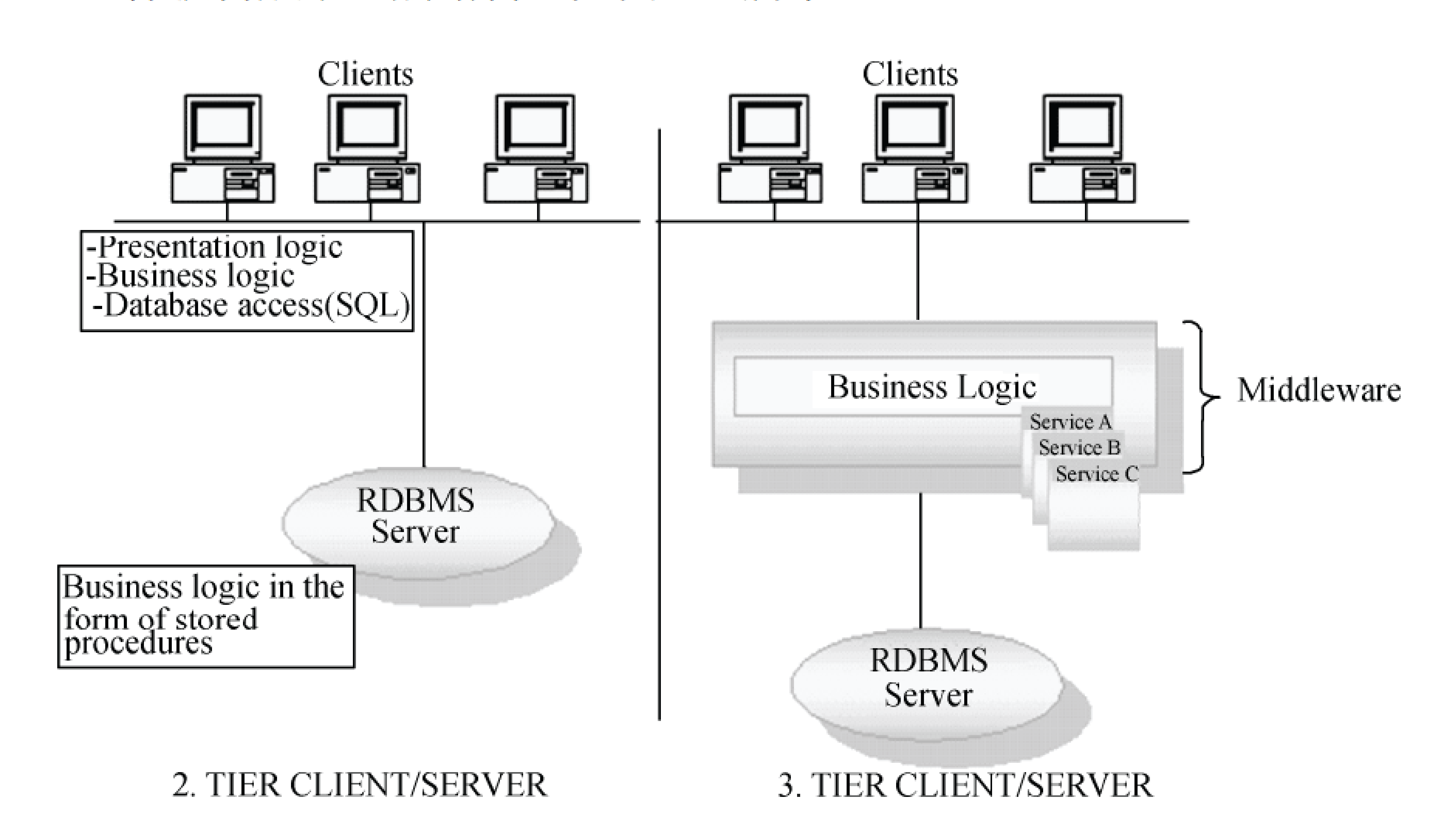


图 1-3

在三层结构中，客户机只处理表示层逻辑（presentation），即显示用户界面、接收用户输入数据、提交交易请求、显示交易处理结果；应用服务器只处理应用逻辑，即负责维护客户机和服务器之间的通信连接，提供命名、通信、事务、安全、并发、持久性、负载均衡等应用服务，做到最大限度地利用系统资源。后台数据库只提供数据服务，即负责数据存储、查询、复制等服务。

表示层、应用逻辑层和数据层在物理分布上是非常灵活的，它们既可以同时分布在同一台主机上，也可以分布在不同的主机上。根据业务需求，可以对应用服务器层进行扩展，这样就形成了多层结构，如图 1-4 所示。

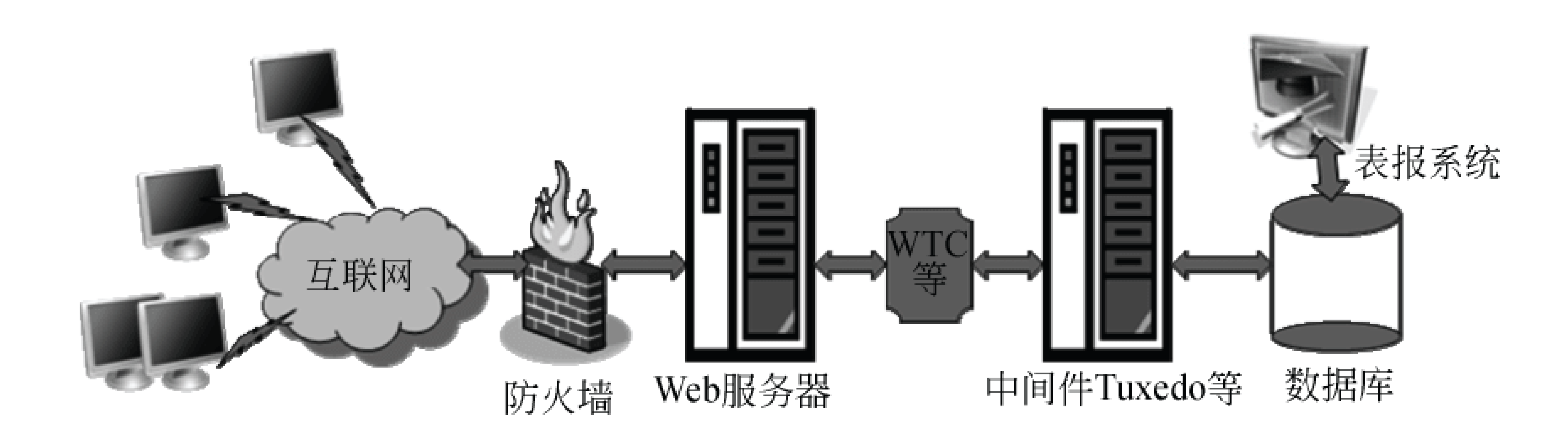


图 1-4

与两层结构相比，三层结构有如下优势。

（1）实现同步、异步、嵌套、转发、会话、事件代理、消息通知、消息队列等丰富的通信机制 。在两层模式中只能实现“请求/应答”式通信，而在三层结构中则很容易实现多种通信方式。

（2）支持分布式事务：分布式的交易需要分布式事务协调器（DTC）来监控事务。在三层结构中，可以把 DTC 部署在中间层上来协调全局事务。而在两层结构中，只能使用数据库提供的单点局部事务，很难进行分布式事务的交易。

（3）资源可以得到充分的利用：应用服务器可以采用多机、多线程、软件集群、软件容错和负载均衡等技术，能够充分利用硬件资源。

（4）灵活性好：与两层结构相比较，三层结构提供的最大灵活性是可扩展性。三层结构中的每一层都是独立的，系统容量可以在项目实施后进行动态调整。

（5）高可靠性：中间件应用服务层可以通过执行队列、执行线程和连接池来控制并发请求，通过集群、负载均衡和容错机制来提供可靠性和可用性，真正做到 7×24 小时不间断提供服务。

（6）可移植性好：商业化的中间件系统一般采用可移植性好的语言来开发，例 C/C++ 和 Java，因此可以广泛地支持当前主流的操作系统。

（7）安全性高：三层结构的安全性首先体现在它实现了客户机表示层和数据层的隔离，有效地回避了前端业务人员不小心对数据造成的永久性修改。还可以在中间应用服务器上部署软件模块实现对用户登陆进行验证、授权和访问控制。并可以通过 SSL、LLE、PKI 等实施验证和加密。

（8）可管理性强：在三层结构中，由于业务逻辑都部署在应用服务器上，因此很容易实现集中式的管理，可以通过操作系统来管理和配置进程、线程、内存、IPC 等资源，通过商业的中间系统来管理和监控连接池、执行队列、执行线程。此外还可以在中间系统上安装 SNMP Agent ，由专业的网管软件来管理和监控。

Tuxedo 是支撑三层结构的强大的中间件。它具备上述所有实用特性，并且性能高，稳定性好。它不仅部署灵活，而且针对不同的部署进行了相应的优化。

3．本地客户端

本地客户端是指与 Tuxedo 服务器在同一台机器上，不用通过网络就可以访问到 Tuxedo 服务的客户端。客户端（进程）和服务器端（进程）通过 IPC（进程间通信）进行通信。当然，本地客户端也可访问其他服务器，这样就成了远程客户端。由于共享内存是实现进程间通信的最快方式，而本地客户端正是通过查询共享内存中公告板（BB）的信息，将请求发送到指定服务进程的消息队列，因此提高了请求服务的速度。

4．远程客户端

WORSTATION CLIENT（远程客户端）：是指需要通过网络才可以访问到 Tuxedo 服务器的客户端。远程客户端与 Tuxedo 服务器不在同一台机器上，它通过网络连接到到 Tuxedo 的代理进程（WSL/WSH），再由代理进程执行服务调用，并将服务响应返回给客户端。这样实现了客户端的灵活部署和与服务器端的隔离，如图 1-5 所示。

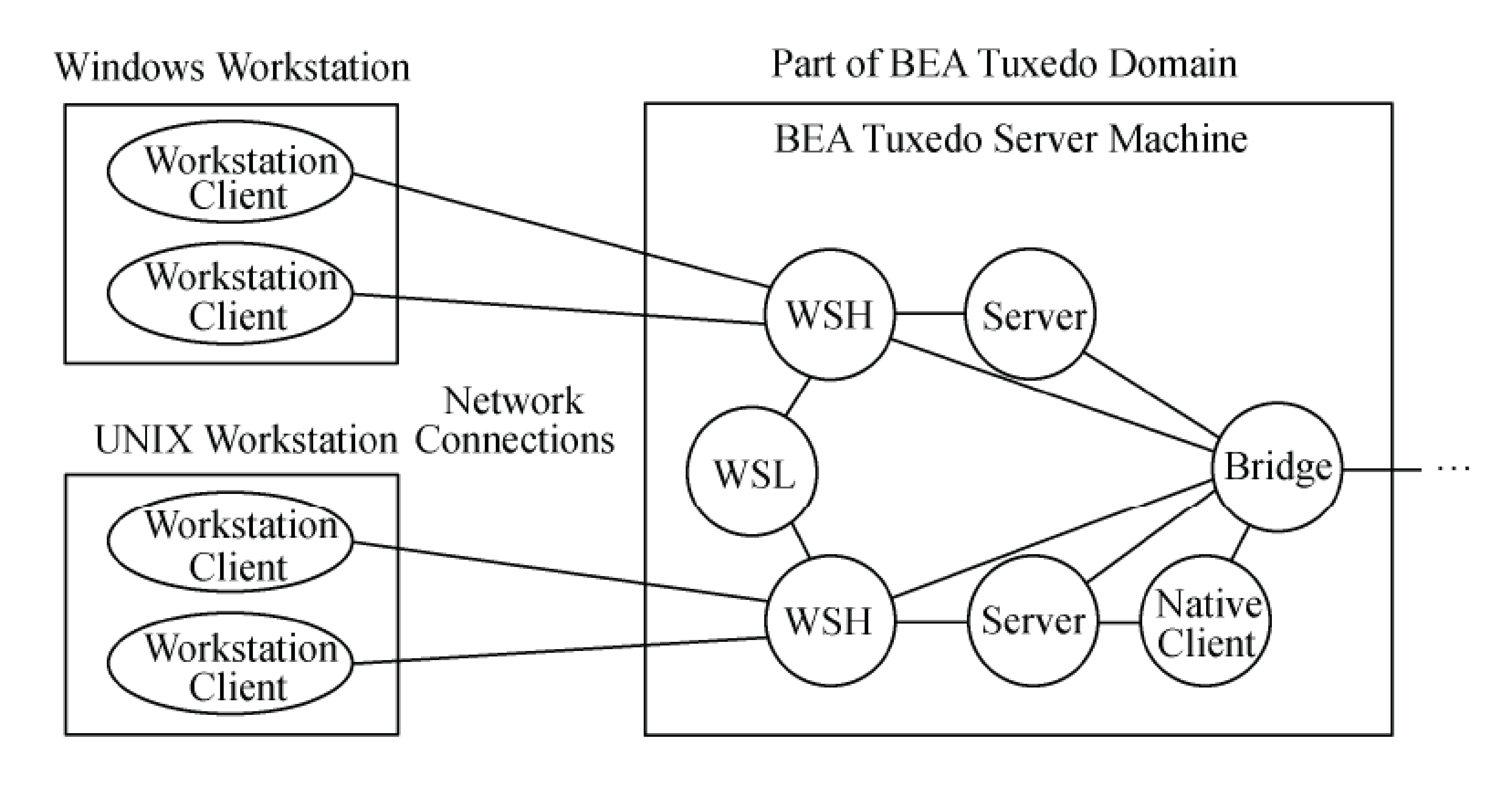


图 1-5

1.4.2 Tuxedo ATMI 体系结构

大多数 Tuxedo 应用都是采用 ATMI（Application-to-Transaction Monitor Interface）来实现运行环境和编程接口。

ATMI 的体系结构如图 1-6 所示。

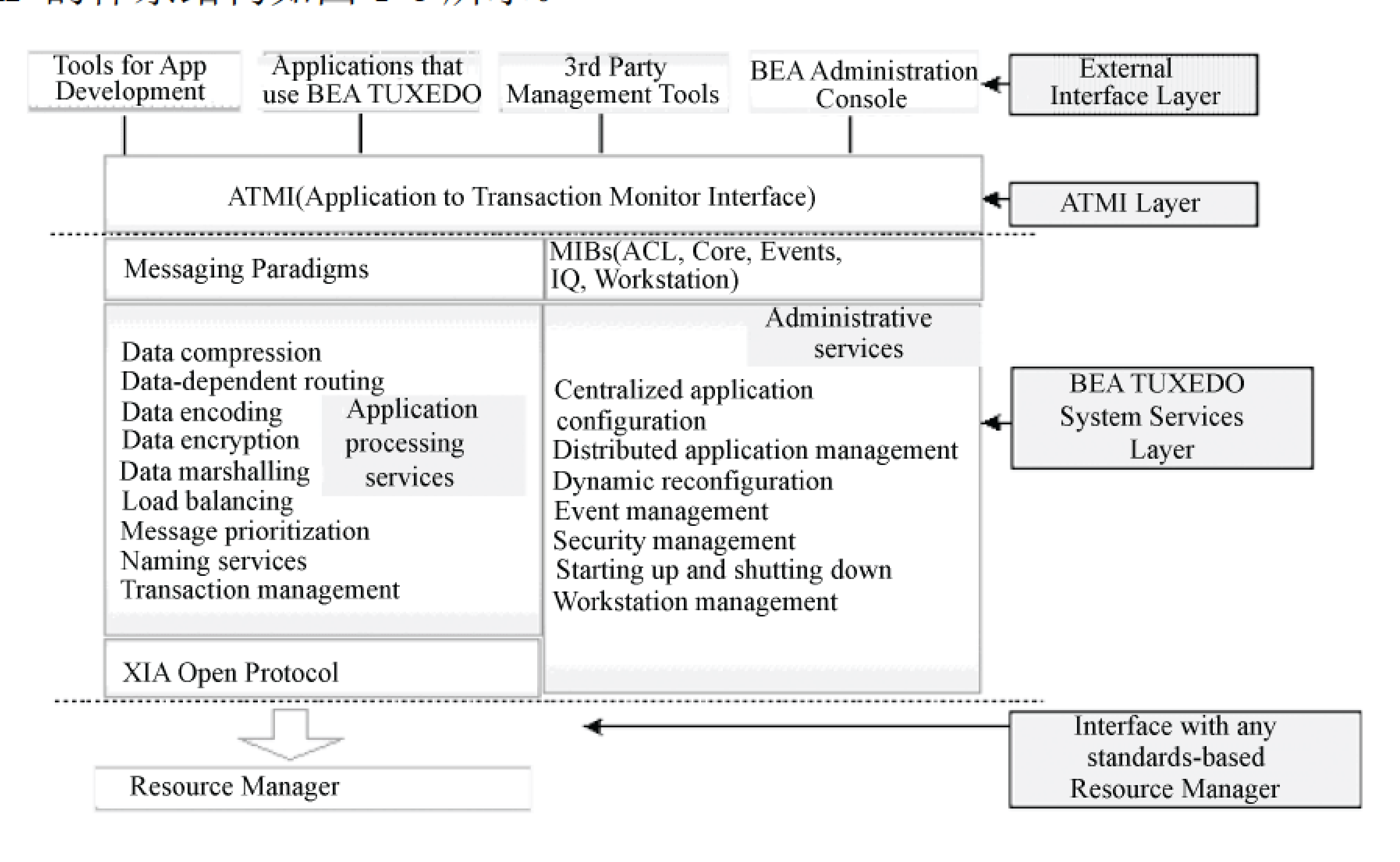


图 1-6

整个结构可以分为外部接口层和 Tuxedo 系统服务层两个部分。外部接口层的基础是 ATMI 层，之上是客户机，包括 Tuxedo 开发工具、Tuxedo 工作站、第三方管理工具和管理控制台。

Tuxedo 系统服务层包括通信模型（messaging paradigms）、管理信息库（MIBs）、应用服务器和管理服务器。通信模型指的是 Tuxedo 客户端与服务器，以及 Tuxedo 服务器与服务器之间传递消息的模式。ATMI 环境支持的通信模型有请求/应答式通信、会话通信、队列通信、事件代理和消息通告。管理信息库为其他应用程序管理和配置 Tuxedo 系统提供了一套编程接口，通过这套接口可以对 Tuxedo 系统进行动态调整和监控。

应用服务为 ATMI 应用程序提供了数据压缩、数据依赖路由、数据编码、数据加密、数据编集、负载均衡、消息优先级、命名服务和事务管理等基础服务。

管理服务为 ATMI 应用程序提供了事件管理、安全管理、事务管理、工作站管理、应用程序运营维护管理等基础服务。

资源管理器（Resource Manager，RM）是一种管理持久数据存储的软件产品。最常见的资源管理器是 DBMS 和消息队列。Tuxedo 的事务系统实现了 X/Open 的 DTP 模型，是一个标准的事务监控器（TP Monitor，TM），它是通过两个阶段提交协议（2PC）来协调参与全局事务的资源管理器。

1. Tuxedo ATMI 的 OLTP 模型

按照服务形式的不同，可以将 C/S 模型分为以数据请求为核心的会话模型和以服务请求为核心的联机事务处理（Online Transaction Processing，OLTP）模型。在以数据请求为核心的模型中，客户机给服务器发送 SQL 指令，服务器给客户机返回数据；在以服务请求为核心的模型中，客户机给服务器发送服务请求，服务器执行业务逻辑并返回给客户机一个响应。

Tuxedo 系统属于以服务请求为核心的 OLTP 模型，其结构如图 1-7 所示。

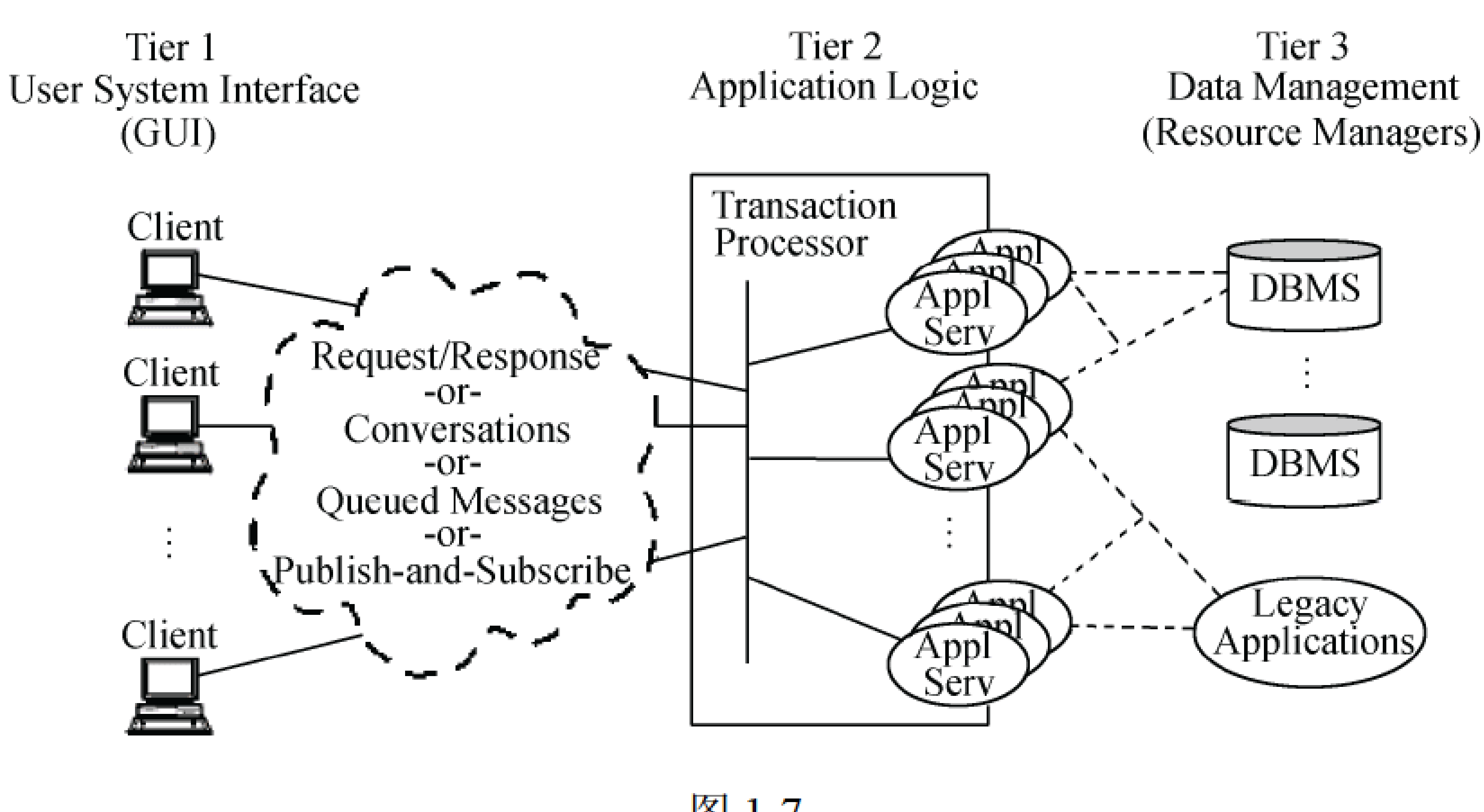


图 1-7

OLTP 最大的特点是并发用户数量大，突发性强，交易时间短，输入输出格式固定。大多数 OLTP 系统都有多个资源管理器来提供数据服务，因此需要借助事务监控器（TP Monitor，TM）来协调分布式事务。

2. Tuxedo ATMI 的命名服务

Tuxedo 使用公告板来提供命名服务。公告板是一块共享内存区域，它保存着服务进程、服务、消息队列、事件、运行环境的配置和统计信息。为了便于快速访问，在运行时系统中，它会被复制到 Tuxedo 系统的每个成员节点上。

如图 1-8 所示，Tuxedo 客户机和服务器可以直接使用名字来调用一个服务，引用一个消息队列和事件，ATMI 命名服务会把这些逻辑请求映射到服务器节点或服务器进程环境内指定的服务实例上。

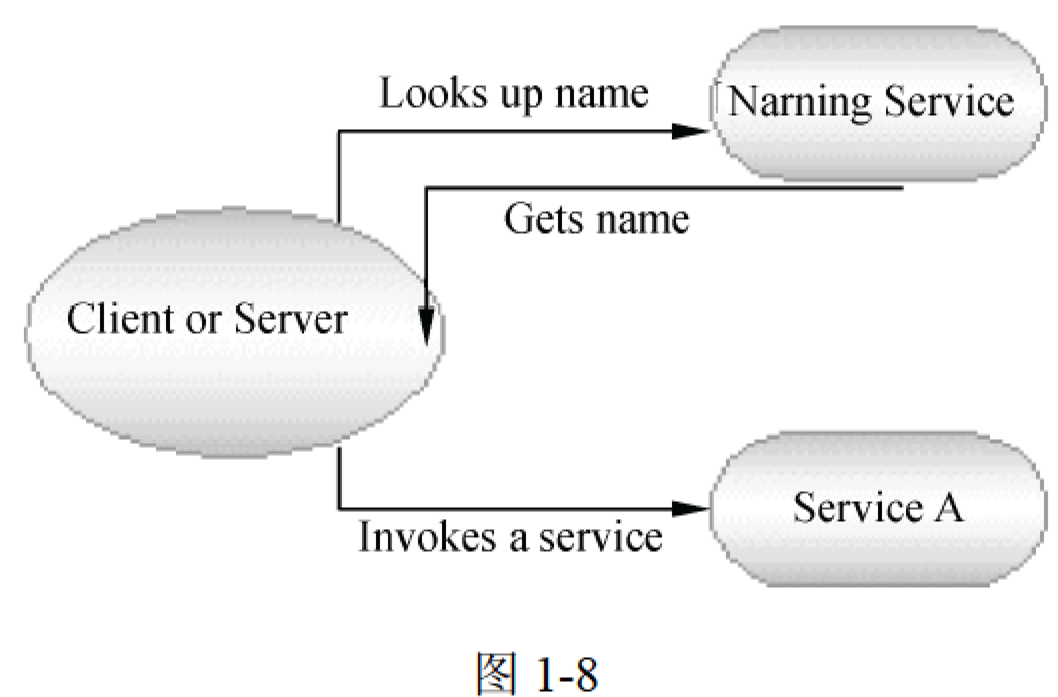


图 1-8

3. Tuxedo ATMI 的消息通信方式

ATMI 环境不但能管理应用服务和事务，而且提供多种 C/S 通信模式：请求/应答式通信（包括同步调用、异步调用、嵌套调用和转发调用）、会话通信、队列通信、事件代理和消息通告。Tuxedo 采用无连接通信，为每一个服务进程分配一个 IPC 消息队列，称为请求队列，同时为每一个客户端分配一个响应队列。客户端的请求放入相应服务器的请求队列，之后服务返回的数据放回客户端的响应队列，如图 1-9 所示。

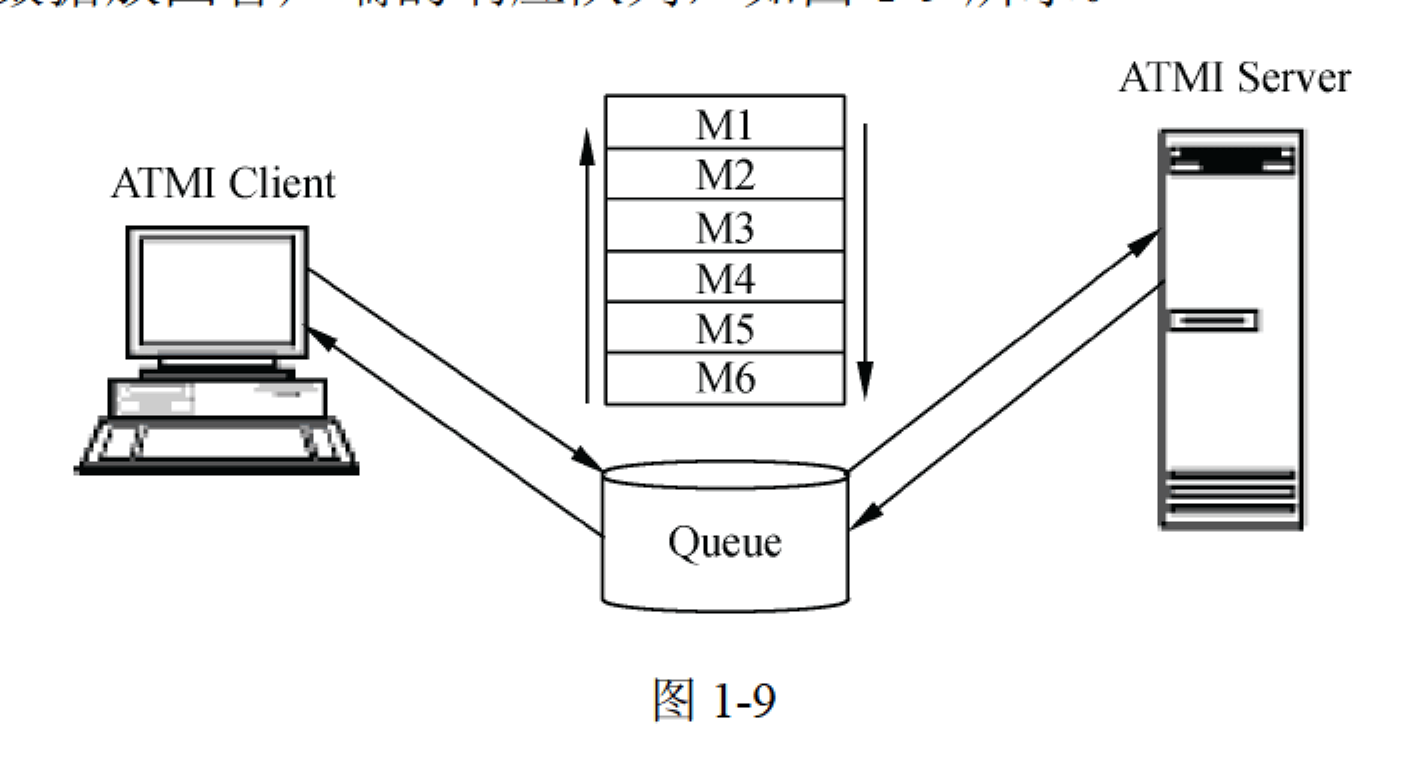


图 1-9

4. Tuxedo ATMI 的消息缓冲区

在 ATMI 编程环境中，客户机和服务器之间使用消息缓冲区来进行数据交换。由于 ATMI 消息缓冲区具有格式化和自描述的特点，因此又称为类型缓冲区（type buffers）。

类型缓冲区克服了平台的差异，为不同系统下的数据表示提供了统一的格式，使 Tuxedo 跨平台进行联机交易和数据转换成为可能，因此它是 ATMI 分布式系统编程环境中最基本、最重要的特征。Tuxedo 支持的类型缓冲区有 STRING、VIEW、CARRAY、FML、XML、MBSTRING。

5. Tuxedo ATMI 消息处理流程

在 ATMI 编程环境中，Tuxedo 客户机和服务器不直接建立通信连接，而是使用面向无连接的 IPC 消息队列来进行数据交换。

消息处理流程如图 1-10 所示。

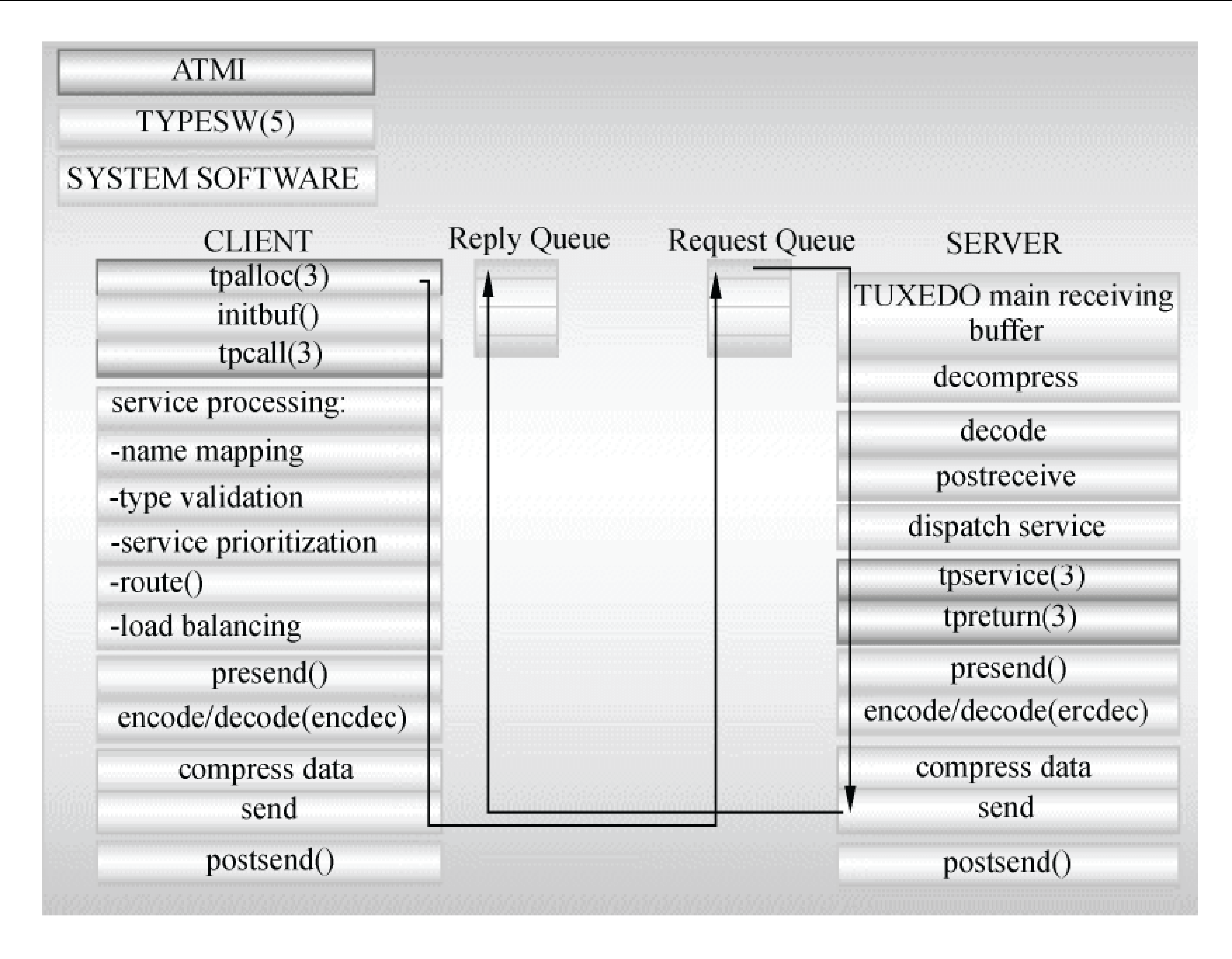


图 1-10

Tuxedo 客户机使用 tpalloc()分配一个请求缓冲区，然后往里面放入请求消息，再执行 tpcall()去调用一个服务。客户机 Tuxedo 系统会根据 tpcall()指定的服务名进行命名映射（name mapping），找出实现这个服务的后台进程的 IPC 消息队列入口，然后进行类型判断（type validation），检查请求消息的缓冲区格式是否是符合服务参数的要求。

如果符合，就从 Tuxedo 服务器的运行系统中读出服务的优先级，并把它绑定到请求消息上（service prioritization）。

在数据依赖路由处理中，客户机 Tuxedo 系统根据路由标准决定把请求消息发送到哪一个后台进程 IPC 消息队列。如果有多个不属于同一个 MSSQ 集合的后台进程可以同时处理这个客户请求，那么，客户机 Tuxedo 系统根据负载均衡（load balancing）的算法，来决定把请求放入哪个后台进程 IPC 消息队列。之后 Tuxedo 系统还可以对请求消息进行编码（encode）、压缩（compress）、事务上下文设置、安全设置等。

最后，客户机 Tuxedo 把请求消息发送到某个服务进程的 IPC 消息队列中。

在 Tuxedo 服务器端，服务进程从 IPC 消息队列中取出来请求消息，经过解压缩（decompress）、解码（decode）之后，交给有名服务去处理，处理结果通过 tpreturn()返回到客户机的 IPC 消息队列中。

1.4.3 Tuxedo CORBA 体系结构

从 Tuxedo 8.0 开始，Tuxedo 提供了 CORBA 分布式对象系统的支持。

Tuxedo CORBA 实现了对象管理组织（OMG）定义的标准，为编写高性能的企业级应用提供了一种基于 CORBA 的解决方案，它为 ORB 模型增加了 OLTP 功能，为 CORBA 分布式对象提供了一个可管理的、集成事务和安全的解决方案。

Tuxedo CORBA 体系结构如图 1-11 所示。

图 1-11

Tuxedo CORBA 体系结构包括如下几个主要部件。

（1）bootstrap Object（引导对象），用于建立和 IIOP Listener/Handler 的连接，并获得 Tuxedo Domain 中的对象引用。

（2）IIOP Listener/Handler（IIOP 监听器/处理器），用于监听和处理 CORBA 客户机发出的请求。

（3）ORB（对象请求代理）通过连接网络上的不同对象的“软件总线”，来提供对象的定位和方法调用。

（4）FactoryFinder 对象，为 Tuxedo CORBA 对象提供生命周期管理服务。

（5）TP Framework（事务处理框架结构），是一个高性能的运行时体系结构，主要特征包括高可用性、集中连接管理、高性能路由连接管理和负载均衡机制。它屏蔽了复杂的 CORBA 接口 ，为快速构建复杂的 CORBA 应用程序提供了编程模型。TP Framework 主要负责初始化 CORBA 服务器，创建对象引用，注册对象工厂，管理对象状态，执行启动和停止操作。

（6）TransactionFactory 和 TransactionCurrent 对象，实现了 CORBA 的对象事务管理（OTM）模型，提供对象事务服务（OTS），它允许客户机参加全局事务，管理着联机交易，保证数据的一致性。OTS 可以对跨越多个编程模型、数据库和应用程序的事务进行管理。

（7）SecurityCurrent 和 principalAuthenticator 对象，实现了 CORBA 的安全服务，为 CORBA 服务机提供验证。在 Tuxedo CORBA 应用环境中，可以通过 SSL 和 LLE 来为 IIOP 提供底层数据加密支持，也可以通过开放安全插件接口（SPI）来整合第三方解决方案。

1. Tuxedo CORBA 的 OLTP 模型

Tuxedo CORBA 把 Tuxedo 强大的事务处理技术和 CORBA 的编程模型有机地结合在一起，把 ORB 模型引入到 OLTP 的应用中，它通过 OTM 来对对象事务进行管理。与 ATMI 环境中的 TM 一样，OTM 连接着后端 RM，提供分布式事务管理、负载均衡和容错机制，为前端提供 OLTP 服务。

Tuxedo CORBA 的 OLTP 模型如图 1-12 所示，从层次结构上来说，和 ATMI 环境的 OLTP 模型是一样的，也是分为用户界面层、应用逻辑层和数据库管理层。不同的是，Tuxedo CORBA 的客户机和服务器之间不再直接通过 TCP/IP 协议进行通信，取而代之的是 IIOP，ATMI 环境中的客户机和服务器的通信通道也被 ORB 所取代。

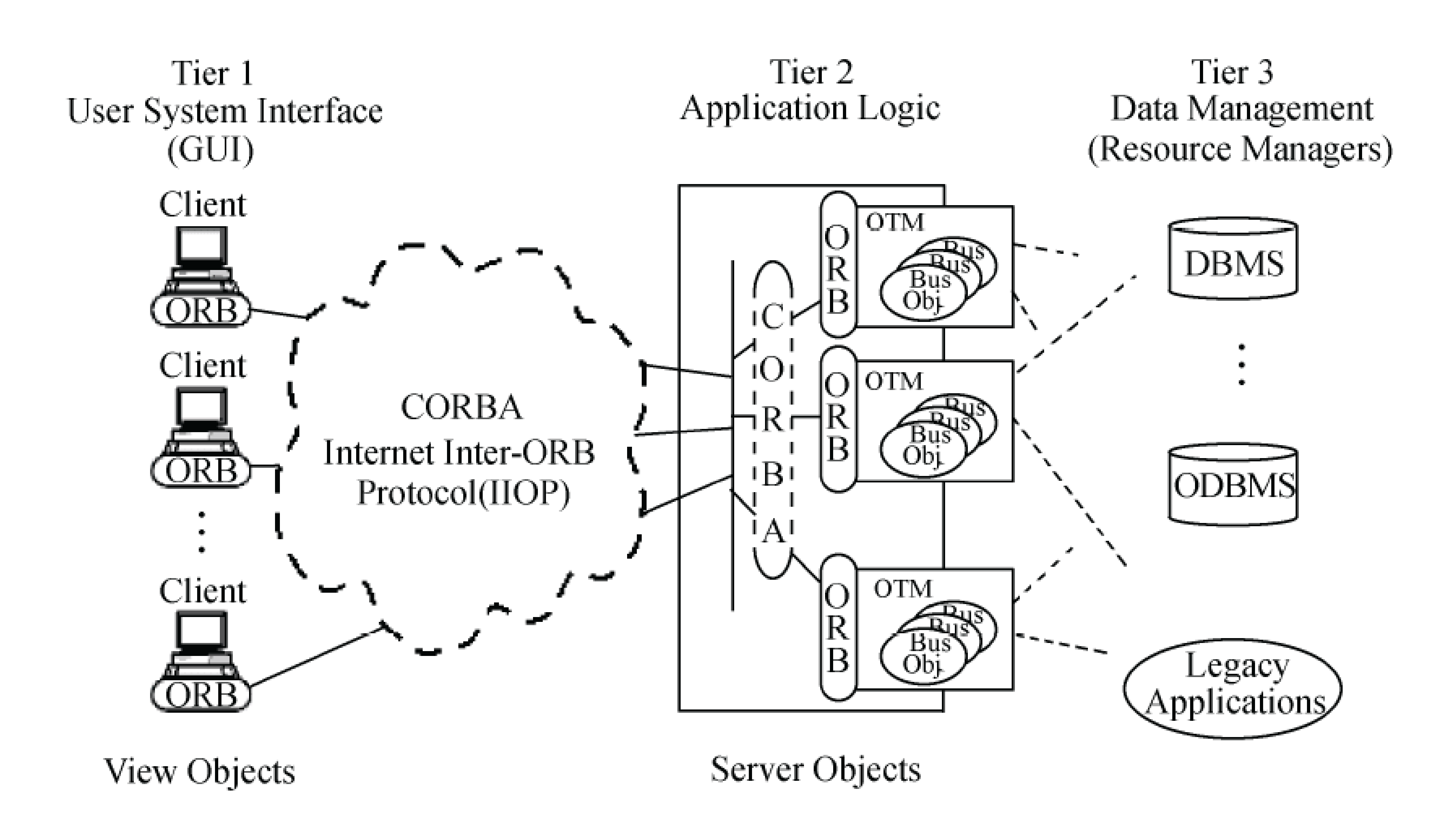


图 1-12

2. Tuxedo CORBA 的 ORB

Tuxedo CORBA 提供了一个更加灵活的方法来开发分布式应用程序，在这个模型中客户机和服务器通过 ORB 进行通信，如图 1-13 所示。

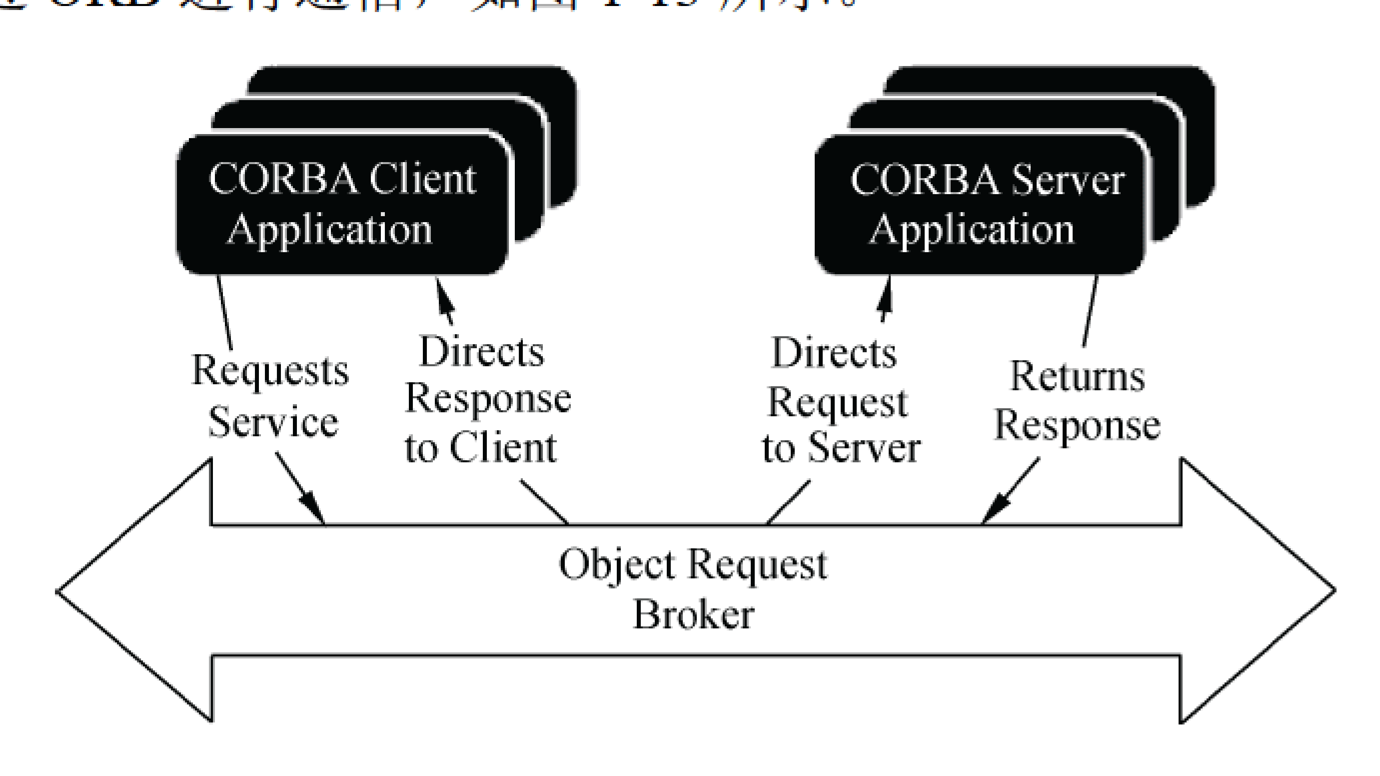


图 1-13

ORB 是介于客户机和服务器之间的中间层，通过 ORB，客户机可以调用服务器上的

对象或者对象中的应用，被调用的对象与客户机不要求在同一台机器上。ORB 在负责通信的同时，也负责寻找适合完成客户请求的对象，在服务器对象完成处理后，还负责把结果返回客户机。客户对象完全可以不关心服务器对象的位置，以及实现它所采用的具体技术和工作平台，甚至不必关心服务器对象的与服务无关的接口信息，这就大大简化了客户程序的工作。

ORB 提供了不同主机和平台之间应用程序的通信和数据转换。从而实现了对象系统之间的无缝连接。

3. Tuxedo CORBA 的命名服务

Tuxedo CORBA 的命名服务为服务器应用提供了一个可以发布对象引用的场所，以便 CORBA 客户机根据逻辑引用名去查找（lookup），并获得位于服务器上的对象。

Tuxedo CORBA 命名服务的对象绑定和查找过程如图 1-14 所示，CORBA 应用服务器通过操作 1 把对象引用（object）和逻辑名（name）绑定（bind）到命名空间（namespace）中，CORBA 客户机通过操作 2 来向命名服务器发出逻辑名解析请求，命名服务通过操作 3 返回解析结果，即逻辑名对应的对象引用。最后，CORBA 客户机再根据得到的对象引用结果去操作 4 调用服务器对象的方法。

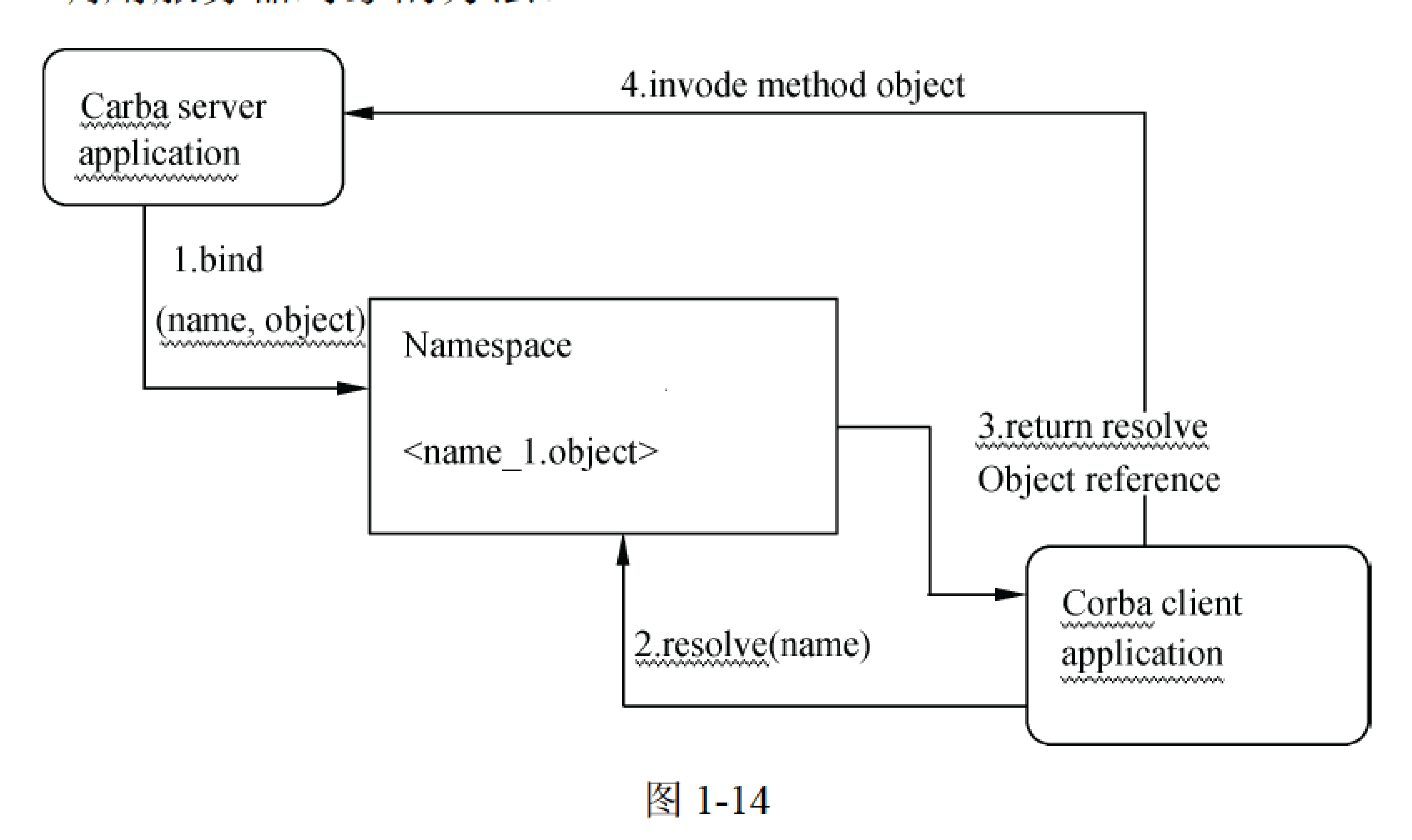


图 1-14

4. Tuxedo CORBA 的通知服务

这里所说的“通知服务”指的就是 Tuxedo CORBA 环境下的事件服务，它与 ATMI 环境中的事件代理服务功能是一样的，即接收事件发布者发布的信息，经过过滤处理后，把它分发到适当的事件订阅者，如图 1-15 所示。

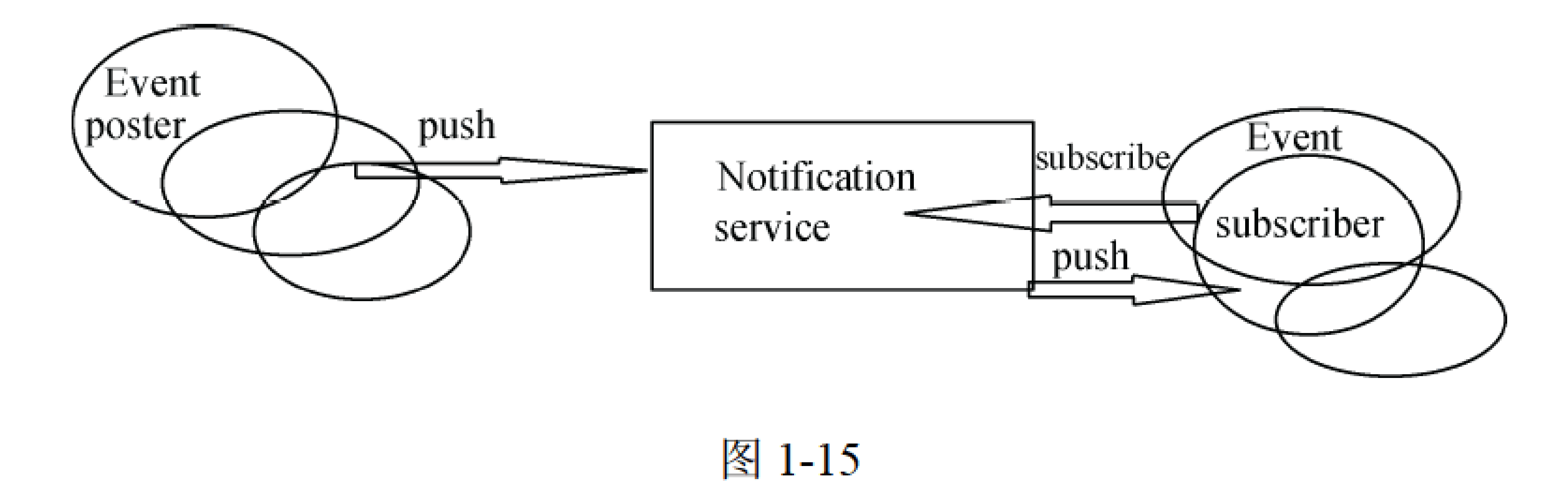


图 1-15

1.4.4　ATMI 与 CORBA 对比

由于 Tuxedo 具有典型的三层结构，因此也具有了三层结构的易于扩展性、可靠性高、安全性好、可管理性好的特点。Tuxedo ATMI 实现了 X/Open 组织定义的分布式事务处理模型，为应用程序提供命名、通信、队列、事务和缓冲区管理等基础服务。Tuxedo CORBA 实现了对象管理组织定义的 CORBA 标准，为编写高性能的企业级应用提供了一种基于 CORBA 的解决方案，它为 ORB 模型增加了 OLTP 功能，为 CORBA 分布式对象系统提供了一个可管理的、集成事务的和安全的解决方案。

ATMI 具有易用性的特点，它可以像 FML 这样具有很强灵活性的缓冲区在客户机和服务器之间传递数据，而 CORBA 只能使用固定的 IDL 来定义接口。CORBA 虽然不好用但是完全是面向对象的。

1.5　Tuxedo 系统的关键特性

1.5.1　名字服务和位置透明性

公告板（BB）为 Tuxedo 应用程序提供了名字服务。为了便于快速访问，它作为共享内存中的一个结构存在，在运行时系统中，公告板会被复制到每个参与的节点上，Tuxedo 系统根据名字信息、配置信息和环境信息自动把请求平衡分派到可用的服务器上，并根据数据内容为客户请求选择路由，为服务器请求选择优先级。

开发人员把应用程序编写成逻辑入口项（称有名服务）函数的调用，Tuxedo 系统会把这些逻辑请求映射到服务器节点及服务器进程上。由于客户机请求的是有名服务，而不是某个具体的后台进程，因此后台进程和服务器的分布对客户机来说是透明的。

1.5.2　强大的 C/S 通信能力

Tuxedo 系统屏蔽了硬件、网络、数据库和操作系统的复杂性，提供了一套简单统一的编程接口，使程序员可以把精力集中在业务逻辑的实现上，而不必为数据库、操作系统和网络通信协议的复杂性、异构性和可靠性担忧。

在 C/S 通信方面，Tuxedo 系统不仅支持“请求/应答”式通信模式（同步、异步、嵌套、转发），而且还支持交易状态的会话通信方式、基于发布/订阅的事件代理方式、基于单播/多播的消息通知方式、基于消息队列的可靠消息存储和转发方式。在消息传递方面，Tuxedo 提供了 CARRAY、STRING、VIEW32、FML32、XML 和 MBSTRING 类型缓冲区来承载消息。此外，Tuxedo 还支持用户自定义消息缓冲区类型，以满足特定数据交换需求。在消息传输过程中，Tuxedo 的高效数据压缩机制平均可以压缩掉 80%数据，大大提高了网络传输效率。

1.5.3 强大的联机交易性能

在 Tuxedo 客户机和服务器之间，以及服务器和服务器之间的通信中，网络上传输的只是一些相对较少的客户机或服务器的请求和服务器的处理结果，而不再是两层结构中客户机和 DBMS 之间的大量 SQL 请求和应答。Tuxedo 能够使多个客户机连接到一个服务器进程，由这个服务器进程统一存取数据库，为客户机的请求服务。这样，数据库为处理连接所需的资源大大减小。

此外，利用 Tuxedo 特有的一些机制也能够极大地提高应用系统的性能。比如，利用异步 RPC 机制实现扇出并行，利用转发机制实现流水线并行，利用 MSSQ 实现多处理并行。所有这些因素使 Tuxedo 应用系统具有极高的性能。

1.5.4 强大的分布式事务协调能力

作为一个强大的 TP Monitor，Tuxedo 协调分布式事务的机制后来形成了 XA 规范。例如，一个客户端发起的事务需要对 4 个数据资源做同步修改，所有的操作都在两个 Tuxedo 服务器的监控下完成，对于客户端来说，只有 4 个操作都成功时，事务才算成功，才能交易，否则就是失败的，必须回滚。

Tuxedo 使用全局事务来跟踪事务参与者，使用两阶段提交协议来协调事务，这样就可确保每个资源管理器都能正确地处理事务的提交和回滚。Tuxedo 还能在网络故障或全局资源死锁时协调全局事务的恢复。

1.5.5 完善的负载均衡机制

Tuxedo 系统使用负载均衡机制把客户请求平均地分布到每一个提供相同服务的后台服务器进程上。负载均衡机制避免了系统中一部分服务器空闲而另一部分服务器却很忙的情况，有效地保证了系统高效的运行。

Tuxedo 系统支持主机级和进程级的负载均衡。如果应用程序分布在多台主机上，则当客户请求到达时，Tuxedo 系统会根据主机的计算能力来分发请求，请求到达某个主机后，Tuxedo 系统会在多个对等的进程之间进行进程级负载均衡。

在默认的情况下，Tuxedo 系统自动地做负载均衡，但用户也可以通过配置负载因子的方法来干预系统的负载平衡调度，Tuxedo 系统会动态地计算每个服务器请求队列中负载因子的总和，然后将请求放进负载最小的服务器队列中。通过配置 MSSQ（Multiple Server Single Queue）可以实现进程级的负载均衡。

1.5.6 数据依赖路由

数据依赖路由（Data Dependent Routing，DDR）是根据请求缓冲区中指定字段的取值范围，来把请求映射到某个服务器组上的算法。

如图 1-16 所示，Server1 和 Server2 是两个完全相同的服务器进程，它们都提供 A 服

务，但是它们部署在不同的服务器中，Server 1 只受理取款低于$500 的交易，Server 2 只受理取款高于$500 的服务，当客户端发出交易请求时，取款低于$500 的交易请求会被分发到 Server 1 去处理，取款高于$500 的请求会被分发到 Server 2 去处理。

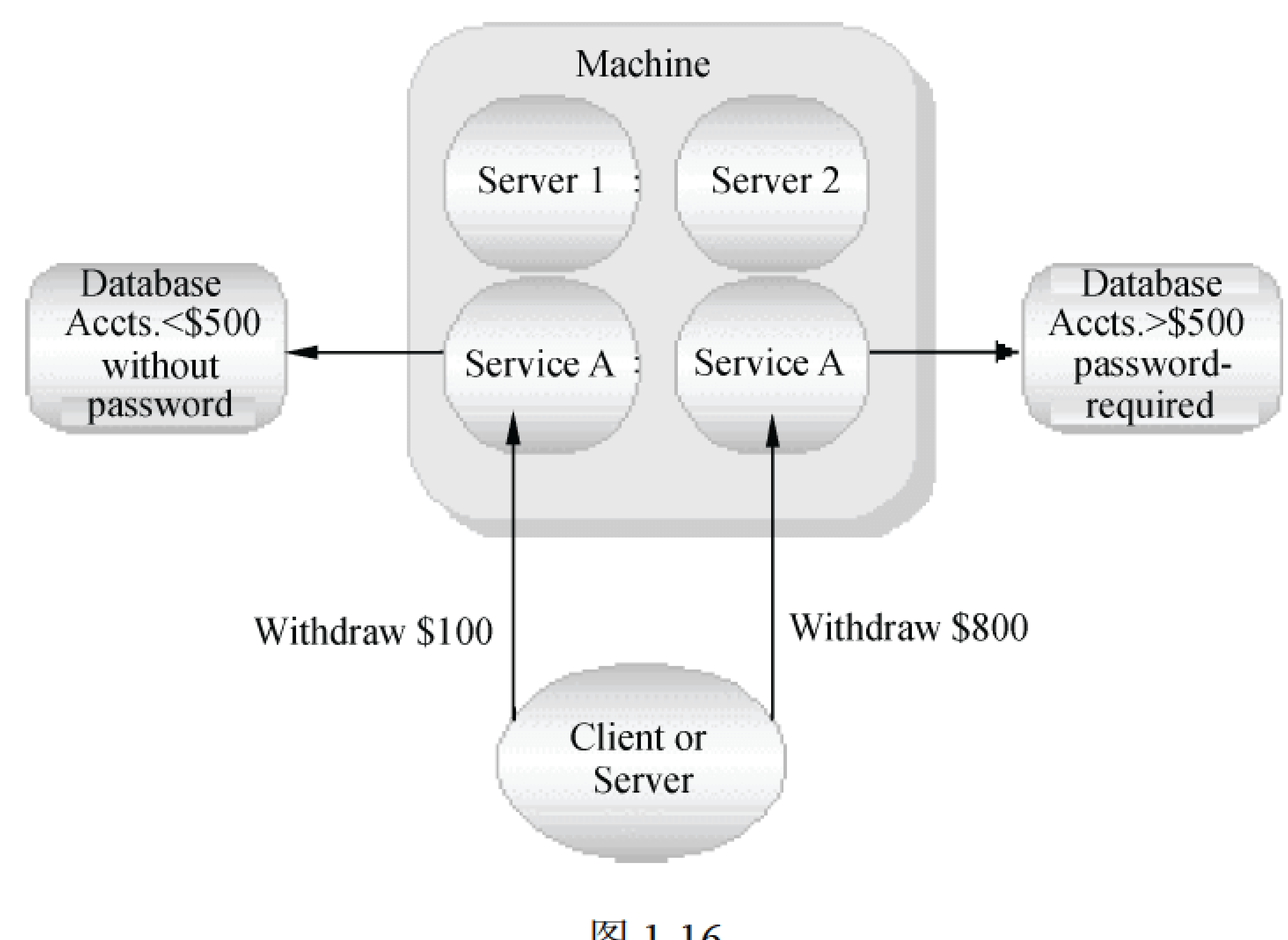


图 1-16

1.5.7　请求的优先级

请求的优先级是 Tuxedo 系统的一个重要特性。在实际应用中，某些服务请求经常需要比其他服务请求有更高的优先级，如航空公司取消订座的优先级要比订座的优先级高，原因不言而喻。

如图 1-17 所示，服务器提供了 3 个服务 A、B、C，优先级分别是 50、50、70。在上一个请求完成时，服务器要从队列中选择下一个请求，这个请求的选取首先是由优先级决定的，而不是根据请求在队列中的位置来决定的，因此第 3 个请求 C 会先出队。为了防止低优先级请求总是得不到处理，每 10 个请求，会按照 FIFO 次序来做一次请求选择。

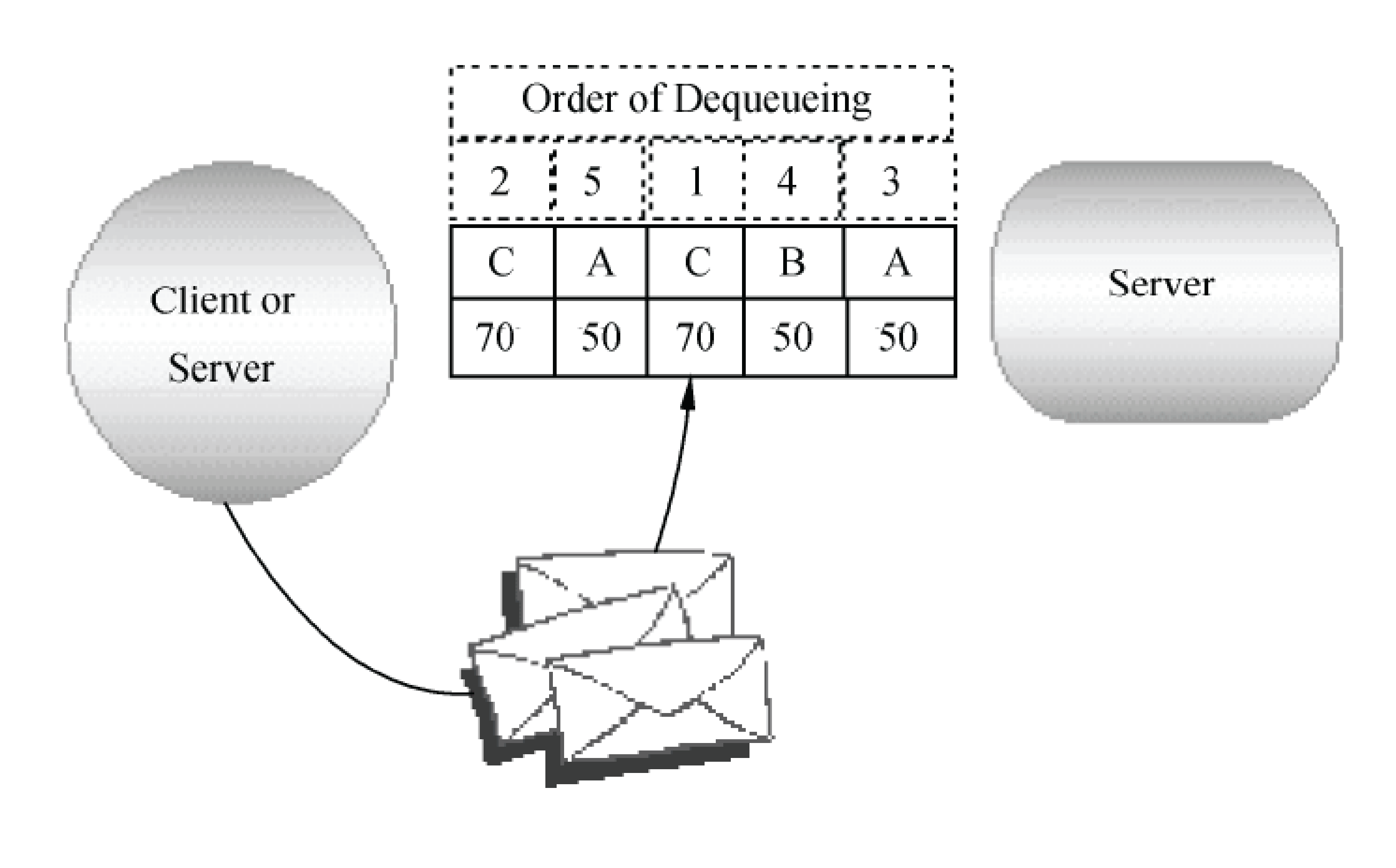


图 1-17

1.5.8 容错和透明故障迁移

保证业务应用持续正常地运行是 Tuxedo 系统的关键特性之一。Tuxedo 通过提供多个备用的服务器组来避免系统中出现单点故障。它的系统部件始终在监视着应用程序、事务、网络和主机状态。

当故障发生时，Tuxedo 系统逻辑上会把故障部件从系统中清除，并且启动故障恢复程序，把请求消息和事务重新路由到系统中的其他部件上去处理。整个过程不会造成服务中断，因此对于客户端工作站来说是完全透明的。

1.5.9 安全性

如图 1-18 所示，Tuxedo 通过一个结构化的安全接口提供验证（authentication）、授权（authorization）和访问控制（access control），该接口允许 Kerberos 或类似的用户自定义验证模块和应用系统集成。用户能用访问控制列表保护服务、队列和事件免遭没有授权的访问。Tuxedo 的安全特性还包括数据加密（encryption）。

Tuxedo 支持两种类型的数据加密，即网络级加密和应用级加密。

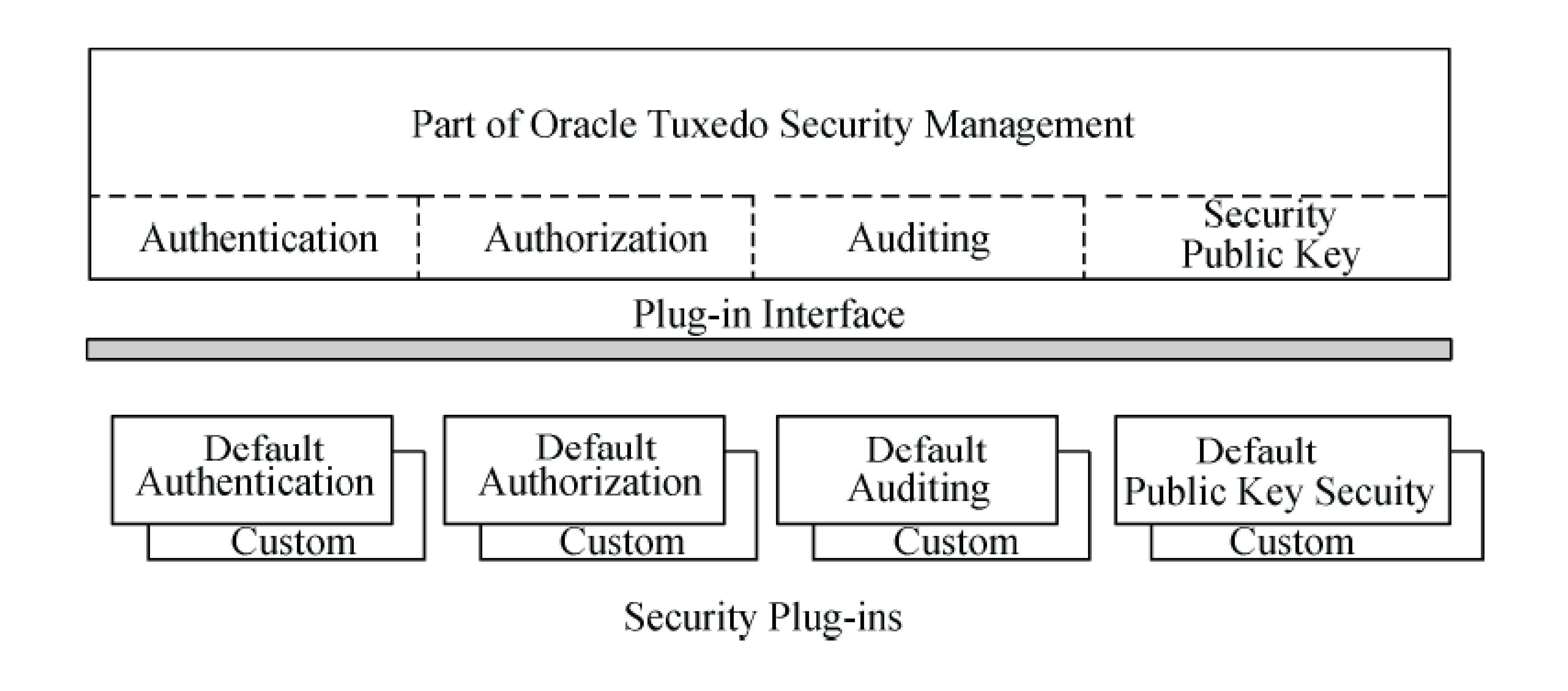


图 1-18

1.5.10 开放性和易用性

Tuxedo 系统具有非常好的开放性，它的核心部件可以运行在所有现在主流的服务器操作系统上。

Tuxedo 的服务器端支持 C/C++、COBOL 开发语言，客户端支持 C/C++、Java、Visual Studio.NET、Delphi、PowerBuilder、Develop/2000 等 RAD 工具，以及 4GL 和 CASE 工具。

Tuxedo 支持很多工业标准，如 X/Open 工业标准、CORBA 分布式标准、轻量级目录访问协议（LADP）等。

Tuxedo 提供简洁 API 使用户程序能够透明地在客户机和服务器之间、服务器和服务器之间进行各种方式的通信，极大地减轻了开发人员的难度和负担。

1.5.11　先进的组织架构

Tuxedo 使用域（Domain）来组织应用系统。一个应用系统可能由单个或者多个 Domain 构成，这些域可以分布在不同的地理位置，域和域之间可以通过局域网或广域网连接在一起，并通过网关实现分布式业务集成。

如果一个应用系统只由一个 Domain 构成，则称为单域应用系统；如果是多个域，则称为多域应用系统。

1.6　Tuxedo 与其他产品横向与纵向的比较

1.6.1　CICS 简介

IBM CICS 是一个商业应用程序服务器。CICS 客户机/服务器功能可以在局域网和广域网络上通过分布和共享资源，管理所有应用程序。它适用于各种系统，从最大的主机系统 S/390 到最小的桌面系统 PC，可以在多种平台之间相互操作并根据企业规模灵活地缩放。

CICS 的多功能性可以确保系统的灵活性。CICS 的用户能够拥有客户机/服务器应用程序灵活互连的能力，无论最终用户和客户处于网络中的何种位置，也无论是在微机还是在大型机中。并且，CICS 还可以连接到很多数据库管理系统上，由于整个网络资源的透明定位，CICS 用户可以从一个应用程序同时访问多个数据库，或同时运行不同数据库上的应用程序。

1．商业能力

CICS 提供了用于关键应用程序的全功能事务处理，其开放式 API 支持开放系统的标准。因此，CICS 用户不必为今后的扩展性担忧，可以将任何需要的开发技术连入系统，也可以根据开发业务所需的用户数目调整系统的规模，将现有的众多应用程序直接移植到 CICS 平台上。

2．客户机/服务器

CICS 建立在客户机/服务器（C/S）的基础上，其应用程序可以根据业务需要分布功能，例如使用 Motif 图形用户界面在客户机工作站上进行显示。

3．服务器支持

CICS 提供了 IBM RISC System/6000 和非 IBM UNIX 工作站的支持，内置的工作负荷管理功能有助于保持对客户机工作站的一致而可预测的响应。

4．客户机支持

CICS 客户机允许用户在台式系统中使用应用程序，并根据业务需要逐步开发显示和分

布功能，从而使工作站的投资发挥作用。

用户可以根据下面 3 个步骤开发客户机应用程序。

（1）模仿 3270 显示器，使位于网络中任何位置的客户机都能够访问 CICS 应用程序和数据。

（2）在客户机的应用程序中增加 GUI 前台，以便能够利用先进的用户界面，使用户获得更高的效率。

（3）开发分布式客户机/服务器应用程序，其表达逻辑运行在客户机上，CICS 应用程序和数据可以在网络中的任何位置。

5．客户机与服务器间的通信

CICS CLIENT 与 CICS SERVER 之间可以采用 TCP/IP、NETBIOS、SNA 或 RPC 协议，其连接方式也是多种多样的，不同的 CICS CLIENT 上的 ECI 应用可以选择以不同的协议连接至不同的 CICS SERVER 上。

6．CICS 与数据库

CICS 与数据库之间是 CLIENT 与 SERVER 的关系，亦即 CICS 通过标准的 XA 接口向数据库提出服务请求并确保该交易的完整性与数据的一致性。

CICS 完整的体系结构保证了 CICS SERVER 与数据库之间连接的灵活性。同一个 CICS SERVER 可以同时与多个 INFORMIX（或其他）数据库建立连接，而且这些数据库可以与 CICS SERVER 位于同一台机器上，也可存在于不同平台的机器上。比如，可以拿一台 RS/6000 作事务处理（CICS）服务器，而用一台 HP/9000 作为 INFORMIX 服务器。不同平台上的 CICS SERVER 可以连接各自不同的数据库。

CICS 还支持工作站或主机上的传统数据文件系统以及 IBM 和非 IBM 的关系型数据库。

1.6.2 Tuxedo 和 CICS 的对比

两种产品都具有事物处理、多种语言支持、安全验证、与多种数据库连通和负载均衡等交易中间件的基本功能。但在易用性和功能的强弱方面有很大差别。

1．产品的产生

CICS 是 IBM 大型机上的一个产品，后来逐步支持其他平台（如 UNIX），因此系统相对比较复杂，不仅编程接口复杂，并且安装完成还要进行复杂的配置。

Tuxedo 由贝尔实验室开发 UNIX 程序员编写，风格比较统一，编程接口简洁，容易理解和安装，有固定的环境变量和配置文件。

2．进程与线程

CICS 为了节省资源一律采用线程，对程序开发人员提高了要求， 一定要使用对线程安全的函数，开发测试环境可能跟实际生产环境存在差异，因此程序具有不可预测性。

Tuxedo 运行用户程序采用进程还是线程由用户决定。采用进程时由于与线程无关，就不用考虑函数的线程安全性问题。但进程相对于线程更耗费资源。

3. 资源情况

CICS 组件多，需要的辅助进程也多，所以支撑其运行所耗费资源相对较多。

Tuxedo 的系统进程少，启动的大部分是应用的进程，资源整合集中利用率高。

4. 程序编写与修改

虽然 CICS 的安装和配置复杂，但安装完之后，调试修改后的应用程序不需要停止服务即可用一条命令更新用户程序到 CICS 系统。

Tuxedo 程序修改后需要重新启动才能生效。

5. 连接多个数据库

CICS 虽然支持连接多个数据库，但使用起来容易出问题。且编程和配置都要针对相应的数据库，并限制启动连接数据库的总数。

Tuxedo 相对容易，它在配置时可以按组连数据库，属于同一组的服务就连相应数据库，与编程无关。

6. 前台编程

CICS、Tuxedo 都支持多种前台开发工具。

第 2 章　Tuxedo 的简单安装和运行

本章将主要讲述 Tuxedo 11gR1 在 Linux32 位平台上的安装和 simpapp 应用程序的配置、部署、运行。

2.1　安装前准备

2.1.1　检查软件包

查看 Tuxedo 相应版本的平台支持列表，检查 Tuxedo 安装包是否和目标系统适应，检查 License 是否到期。

2.1.2　必备的硬件和软件

在将 Tuxedo 安装到 UNIX 平台之前需要检查下列资源。

（1）确定系统有足够的硬盘和内存空间用于安装和使用 Tuxedo。

（2）使用 Tuxedo 平台列表中支持的 C/C++/COBOL/Java 编译器。

（3）调整相应的 IPC 资源（请参考后续章节 3.4）。

在将 Tuxedo 安装到 Windows 系统之前，需要检查下列资源。

（1）确定系统是否有足够的硬盘和内存空间用于安装和使用 Tuxedo。

（2）Windows 的 Administrator 权限。

（3）使用 Tuxedo 平台列表中支持的 C++/COBOL/Java 编译器。

（4）调整相应的 TUXIPC 资源（请参考后续章节 3.4）。

2.1.3　如何获得安装介质及文档

获得安装介质方式可以通过以下途径。

（1）官方网站：www.oracle.com 可以下载一个 Tuxedo 最新版本的试用版。

（2）联系 oracle 销售部门。

2.1.4　Tuxedo 许可证

如果是在官方网站下载的试用版本，那么许可证会发到用户的注册邮箱中。如果是联系销售部门，得到的是安装光盘，那么用户的许可证会在安装光盘中有附带。

2.2　快速安装

2.2.1　Tuxedo 环境要求

Tuxedo 完全安装时，磁盘的使用一般在 120MB 到 380MB 之间，临时空间的使用在 20MB～130MB 之间，不同的平台差异比较大。但是在当前大多数服务器上这点空间可以完全忽略不计。

Tuxedo 运行时对内存的需求也不大，通常在 128MB～256MB 之间，每启动一个额外的进程大约用掉 0.5MB 到 1MB 的内存空间，该要求通常也是可以满足需求的。

Tuxedo 对 CPU 的数量和主频没有太大要求。增加 CPU 处理能力可以提高 Tuxedo 的最大处理性能。

Tuxedo 需要的软件主要就是 C/C++/COBOL/Java 编译器，根据需求安装其中一个就可以。还可以在开发机器上编译再搬到生产机器上运行。

2.2.2　内核参数的调整

为了使 Tuxedo 正常高效地运行，在 Tuxedo 安装前或者是安装后需要对操作系统内核参数进行调整。（请参考后续章节 3.4）

2.2.3　进行 Tuxedo 安装

Tuxedo 的安装有 3 种方式：图形化安装，命令安装和无人值守安装。图 2-1 至图 2-11 所示为在 Linux 中命令安装的方法。

首先把 Tuxedo 的安装包上传到服务器，给予执行权限：

```
-rwxr-xr-x 1 bea bea 94294560 02-10 13:13 tuxedo111120_32_Linux_01_x86.bin
[bea@landingbj ~]$
```

图 2-1

执行安装命令：

```
[bea@landingbj ~]$ ./tuxedo111120_32_Linux_01_x86.bin -i console
```

图 2-2

开始安装：

```
[bea@landingbj ~]$ ./tuxedo111120_32_Linux_01_x86.bin -i console
Preparing to install...
Extracting the JRE from the installer archive...
Unpacking the JRE...
Extracting the installation resources from the installer archive...
Configuring the installer for this system's environment...

Launching installer...

Preparing CONSOLE Mode Installation...

===============================================================================
Choose Locale...
----------------

    1- English

CHOOSE LOCALE BY NUMBER:
```

图 2-3

选择语言 1 后继续：

```
CHOOSE LOCALE BY NUMBER: 1
===============================================================================
Tuxedo 11.1.1.2.0                 (created with InstallAnywhere by Macrovision)
-------------------------------------------------------------------------------




===============================================================================
Introduction
------------

InstallAnywhere will guide you through the Tuxedo 11.1.1.2.0 installation.


It is strongly recommended that you quit all programs before continuing with
this installation.

Enter "next' to proceed to the next screen. Enter "back" to modify the previous
screen.

You may cancel this installation at any time by typing "quit".


WARNING: "Quitting" creates an incomplete  Tuxedo 11.1.1.2.0 installation.You
must re-install  Tuxedo 11.1.1.2.0. For more information, see "Preparing to
Install the Oracle Tuxedo System" in the  Tuxedo 11.1.1.2.0 Installation Guide.


PRESS <ENTER> TO CONTINUE: Command.run(): process completed before monitors could start.
Command.run(): process completed before monitors could start.
```

图 2-4

选择是完全安装还是部分安装：

```
===============================================================================
Choose Install Set
------------------

Please choose the Install Set to be installed by this installer.

  ->1- Full Install
    2- Server Install
    3- Full Client Install
    4- Jolt Client Install
    5- ATMI Client Install
    6- CORBA Client Install

    7- Customize...

ENTER THE NUMBER FOR THE INSTALL SET, OR PRESS <ENTER> TO ACCEPT THE DEFAULT
   :
```

图 2-5

该项可以根据需要选择，选择 1 完全安装：

```
ENTER THE NUMBER FOR THE INSTALL SET, OR PRESS <ENTER> TO ACCEPT THE DEFAULT
   : 1



===============================================================================
Choose Oracle Home
------------------


    1- Create new Oracle Home

Enter a number: 
```

图 2-6

选择 1 后填写安装路径：

```
Specify a new Oracle Home directory: /home/bea/tuxedo



===============================================================================
Choose Product Directory
------------------------


    1- Modify Current Selection (/home/bea/tuxedo/tuxedo11gR1)
    2- Use Current Selection (/home/bea/tuxedo/tuxedo11gR1)

Enter a number: 
```

图 2-7

此处可以修改刚才的路径也可以选择应用现在的路径：

```
Enter a number: 2
Install Samples (Y/N): y



===============================================================================
Pre-Installation Summary
------------------------

Please Review the Following Before Continuing:

Product Name:
    Tuxedo 11.1.1.2.0

Install Folder:
    /home/bea/tuxedo/tuxedo11gR1

Link Folder:
    /home/bea

Disk Space Information (for Installation Target):
    Required:  209,943,511 bytes
    Available: 14,370,181,120 bytes

PRESS <ENTER> TO CONTINUE: 
```

图 2-8

如选择应用当前的路径后继续安装，确认后继续：

```
===============================================================================
Ready To Install
----------------

InstallAnywhere is now ready to install Tuxedo 11.1.1.2.0 onto your system at
the following location:


   /home/bea/tuxedo/tuxedo11gR1

PRESS <ENTER> TO INSTALL:



===============================================================================
Installing...
-------------

 [==================|==================|==================|==================]
 [------------------|-------------
```

图 2-9

开始安装完成后输入 tlisten 密码：

```
===============================================================================
Configure tlisten Service
------------------------

Password:
Verify Password:
Password Accepted! Please wait...
```

图 2-10

是否安装 SSL（为简单计，选择“否”）：

```
===============================================================================
SSL Installation Choice.
------------------------

Would you like to install SSL Support?

  ->1- Yes
    2- No

ENTER THE NUMBER FOR YOUR CHOICE, OR PRESS <ENTER> TO ACCEPT THE DEFAULT:
   : 2



===============================================================================
Installation Complete
---------------------

Congratulations. Tuxedo 11.1.1.2.0 has been successfully installed to:


   /home/bea/tuxedo/tuxedo11gR1

PRESS <ENTER> TO EXIT THE INSTALLER:
```

图 2-11

单击“回车”按钮安装完成！

2.2.4 兼顾需要 License 的版本

下载 License 的时候会提示用户下载一个试用版的 License，否则会无法顺利完成。

2.3　部署应用（simpapp 例子）

以下就以 Tuxedo 中自带的一个简单的例子（simpapp）演示一个应用的部署过程。这个例子只是简单地把输入的字母转换成大写字母输出到控制台上。通过它可以了解 Tuxedo 应用的基本部署、编译、运行过程。

一个应用要成功部署，需要以下几个步骤：

2.3.1　修改配置文件

首先修改环境变量文件 tux.env，要在其中加上几个重要的变量，分别是应用的路径 APPDIR 和配置文件 TUXCONFIG 的路径。

如图 2-12 所示：

```
LANG=C; export LANG
APPDIR=/home/bea/tuxedo/simpapp; export APPDIR
TUXCONFIG=$APPDIR/tuxconfig; export TUXCONFIG

[bea@caochun simpapp]$
```

图 2-12

然后修改配置文件 ubbsimple，进行以下修改，如图 2-13 所示：

```
#IPCKEY         123456

DOMAINID        simpapp
MASTER          simple
MAXACCESSERS    10
MAXSERVERS      5
MAXSERVICES     10
MODEL           SHM
LDBAL           N

*MACHINES
DEFAULT:
                APPDIR="<Replace with the current directory pathname>"
                TUXCONFIG="<Replace with your TUXCONFIG Pathname>"
                TUXDIR="<Directory where TUXEDO is installed>"
#Example:
#               APPDIR="/home/me/simpapp"
#               TUXCONFIG="/home/me/simpapp/tuxconfig"
#               TUXDIR="/usr/tuxedo"

<Machine-name>  LMID=simple

#Example:
#beatux         LMID=simple
```

图 2-13

从上到下分别修改 IPC 资源的信号量键值设置，APPDIR 是应用路径设置，TUXCONFIG 是 tuxconfig 二进制文件产生的路径，TUXDIR 为 Tuxedo 安装目录，machine-name 为本机 hostname，其他的暂时不需要修改。

保存退出。

2.3.2　加载配置文件

执行. ./tux.env 设置好环境变量后，执行 tmloadcf ubbsimple。

如图 2-14 所示：

```
[root@landingbj simpapp]# . ./tux.env
[root@landingbj simpapp]# tmloadcf ubbsimple
```

图 2-14

这样就产生一个二进制的 tuxconfig 文件，到这一步，配置文件便制作好了。

2.3.3　启动 Tuxedo

启动 Tuxedo 相当简单，可以用 tmboot –y 就直接启动了。
如图 2-15 所示：

```
[root@landingbj simpapp]# tmboot
Boot all admin and server processes? (y/n): y
Booting all admin and server processes in /root/simpapp/tuxconfig
INFO: Oracle Tuxedo, Version 11.1.1.2.0, 32-bit, Patch Level (none)

Booting admin processes ...

exec BBL -A :
        process id=16603 ... Started.

Booting server processes ...

exec simpserv -A :
        process id=16604 ... Started.
2 processes started.
```

图 2-15

当然停止 Tuxedo 也很简单，直接用 tmshutdown –y。
如图 2-16 所示：

```
[root@landingbj simpapp]# tmshutdown -y
Shutting down all admin and server processes in /root/simpapp/tuxconfig

Shutting down server processes ...

        Server Id = 1 Group Id = GROUP1 Machine = simple:       shutdown succeeded

Shutting down admin processes ...

        Server Id = 0 Group Id = simple Machine = simple:       shutdown succeeded
2 processes stopped.
```

图 2-16

2.3.4　相关的日志文件

Tuxedo 启动/终止等系统日志记录在 ULOG.<日期>文件中。用户也可以通过编程将用户日志记录在该文件中。

2.4　编译和运行

2.4.1　编译程序

编译应用代码：分别是客户端代码 simpcl.c 和服务器端代码 simpserv.c。

分别用命令：

```
buildclient -o simpcl -f simpcl.c
buildserver -o simpserv -f simpserv.c -s TOUPPER
```

这样就产生了可以执行的客户端和服务器端代码。

2.4.2　运行程序

执行客户端也相当简单。
如下：

```
[bea@landingbj simpapp]$ ./simpcl  helloword
HELLOWORD   (#输出大写的 helloword)
```

2.5　卸载 Tuxedo

在 Windows 平台上可以通过控制面板中的“添加/删除程序”来完成卸载。
在 UNIX 平台中可以通过执行 uninstaller 子目录中的卸载脚本来完成卸载。
例如：

```
[bea@landingbj uninstaller]$ ./Uninstall Tuxedo 11.1.1.2.0 -i console
(# 字符界面卸载)
```

第 2 篇

基　础　篇

第 3 章　OLTP 基本知识

当今的数据处理大致可以分成两大类：OLTP（Online Transaction Processing）和 OLAP（Online Analytical Processing）。

联机事务处理系统（OLTP）也称为面向交易的处理系统，其基本特征是顾客的原始数据可以立即传送到计算中心进行处理，并在很短的时间内给出处理结果。这样做的最大优点是可以即时地处理输入的数据，及时地回答，也称为实时系统（real time system）。

衡量联机事务处理系统的一个重要指标是系统性能，具体体现为实时响应时间（response time），即用户在终端上送入数据之后，到计算机对这个请求给出答复所需要的时间。

而 OLAP 则是数据仓库系统的主要应用，支持复杂的分析操作，侧重决策支持，并且提供直观易懂的查询结果，对数据的及时性没有 OLTP 要求的那么高。

3.1　三层或多层 C/S 架构

在软件体系架构中，有 C/S 和 B/S 两大架构。

C/S（Client/Server）结构，即客户机和服务器结构，通过它可以充分利用两端硬件环境的优势，将任务合理分配到 Client 端和 Server 端来实现，降低了系统的通讯开销。

B/S（Browser/Server）结构即浏览器和服务器结构。它是随着 Internet 技术的兴起，对 C/S 结构的一种变化或者改进的结构。在这种结构下，用户工作界面是通过浏览器来实现的，极少部分事务逻辑在前端（Browser）实现，但是主要事务逻辑在服务器端（Server）实现，形成所谓三层（3-tier）结构。这样就大大简化了客户端电脑载荷，减轻了系统维护与升级的成本和工作量，降低了用户的总体成本。

传统的 C/S 结构为二层结构，它的局限性体现为它是单一服务器且以局域网为中心的，所以难以扩展至大型企业广域网或 Internet。

（1）受限于供应商。

（2）软、硬件的组合及集成能力有限。

（3）难以管理大量的客户机。

因此，三层 C/S 结构应运而生。三层 C/S 结构是将应用功能分成表示层、功能层和数据层三部分。其解决方案是对这三层进行明确分割，并在逻辑上使其独立。原来的数据层作为 DBMS 已经独立出来，所以关键是要将表示层和功能层分离成各自独立的程序，并且还要使这两层间的接口简洁明了。

表示层是应用的用户接口部分，它担负着用户与应用间的对话功能。它用于检查用户从键盘等输入的数据，显示应用输出的数据。为使用户能直观地进行操作，一般使用图形用户接口（GUI），操作简单、易学易用。在变更用户接口时，只需改写显示控制和数据检

查程序，而不影响其他两层。检查的内容也只限于数据的形式和值的范围，不包括有关业务本身的处理逻辑。

图形界面的结构是不固定的，这便于以后能灵活地进行变更。例如，在一个窗口中不是放入几个功能，而是按功能分割窗口，以便使每个窗口的功能简洁单纯。在这层的程序开发中主要是使用可视化编程工具。

功能层相当于应用的本体，它将具体的业务处理逻辑编入程序中。例如，在制作订购合同时要计算合同金额，按照定好的格式配置数据、打印订购合同，而处理所需的数据则要从表示层或数据层取得。表示层和功能层之间的数据交换要尽可能简洁。例如，用户检索数据时，要设法将有关检索要求的信息一次性传送给功能层，而由功能层处理过的检索结果数据也一次性传送给表示层。在应用设计中，一定要避免“进行一次业务处理，在表示层和功能层间进行多次数据交换”的笨拙设计。

通常，在功能层中包含有：确认用户对应用和数据库存取权限的功能以及记录系统处理日志的功能。这层的程序多半是用可视化编程工具开发的，也有使用 COBOL 和 C 语言的。

数据层就是 DBMS，负责管理对数据库数据的读写。DBMS 必须能迅速执行大量数据的更新和检索。现在的主流是关系型数据库管理系统（RDBMS）。因此，一般从功能层传送到数据层的请求大都使用 SQL 语言。

3.2　事务的概念

3.2.1　什么是事务

事务（Transaction）是一组逻辑上相关联的操作，这些操作要么全都成功执行，要么全都不执行。事务所含的操作可以分布在不同的程序甚至不同的机器上。

事务应该具有 4 个属性：原子性、一致性、隔离性、持久性。这 4 个属性通常称为 ACID 特性。

- **原子性（atomicity）** 一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做。
- **一致性（consistency）** 事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。
- **隔离性（isolation）** 一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
- **持久性（durability）** 也称永久性（permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的，除非有新的事务改变它。接下来的其他操作或故障不应该对其有任何影响。

3.2.2　什么是全局事务

所谓全局事务，是指分布式事务处理环境中，多个数据库可能需要共同完成一个工作，

这个工作即是一个全局事务，例如，一个事务中可能更新几个不同的数据库。对数据库的操作发生在系统的各处，但必须全部被提交或回滚。此时一个数据库对自己内部所做操作的提交不仅依赖本身操作是否成功，还要依赖与全局事务相关的其他数据库的操作是否成功，如果任一数据库的任一操作失败，则参与此事务的所有数据库所做的所有操作都必须回滚。

一般情况下，某一数据库无法知道其他数据库在做什么，因此，在一个分布式事务处理（Distributed Transaction Processing，DTP）环境中，交易中间件是必需的，由它通知和协调相关数据库的提交或回滚。而一个数据库只将其自己所做的操作（可恢复）影射到全局事务中。

3.2.3 XA 规范

X/Open 组织（即现在的 Open Group）定义了分布式事务处理模型。X/Open DTP 模型（1994）包括应用程序（AP）、事务管理器（TM）、资源管理器（RM）、通信资源管理器（CRM）4 部分。一般情况下，常见的事务管理器（TM）是交易中间件，常见的资源管理器（RM）是数据库，常见的通信资源管理器（CRM）是消息中间件。

XA 就是 X/Open 为 DTP 定义的交易中间件与资源管理器（如数据库）之间的接口规范（即接口函数），交易中间件用它来通知数据库事务的开始、结束以及提交、回滚等。XA 接口函数由数据库厂商提供。

通常情况下，交易中间件与数据库通过 XA 接口规范，使用两阶段提交来完成一个全局事务，XA 规范的基础是两阶段提交协议，如图 3-1 所示。

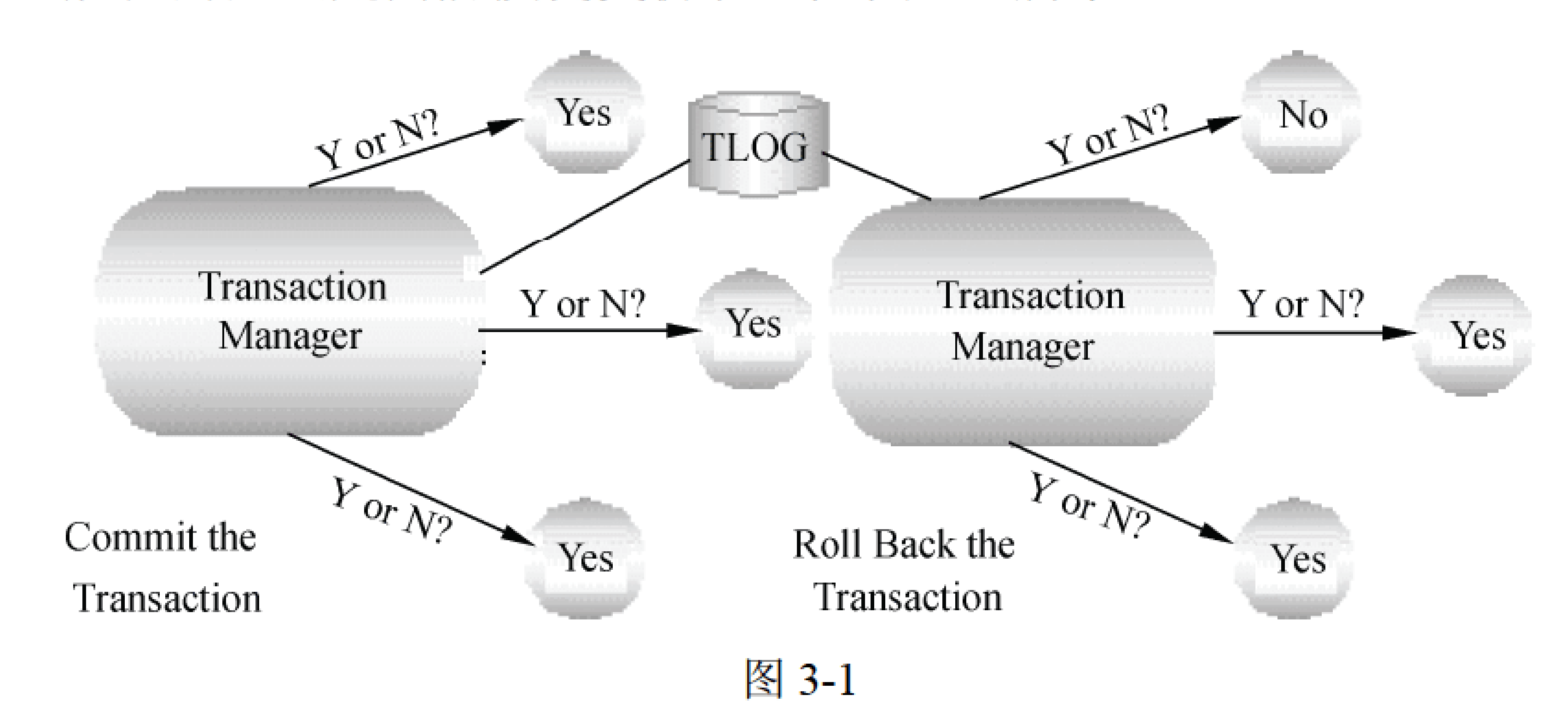


图 3-1

在第一阶段，交易中间件请求所有相关数据库准备提交（预提交）各自的事务分支，以确认是否所有相关数据库都可以提交各自的事务分支。当某一数据库收到预提交后，如果可以提交属于自己的事务分支，则将自己在该事务分支中所做的操作固定记录下来，并给交易中间件一个同意提交的应答，此时将不能再在该事务分支中加入任何操作，但此时数据库并没有真正提交该事务，数据库对共享资源的操作还未释放（处于上锁状态）。如果由于某种原因数据库无法提交属于自己的事务分支，它将回滚自己的所有操作，释放对共享资源上的锁，并返回给交易中间件失败应答。

在第二阶段，交易中间件审查所有数据库返回的预提交结果，如果所有数据库都可以提交，交易中间件将要求所有数据库做正式提交，这样该全局事务被提交。而如果有任一数据库预提交返回失败，交易中间件将要求所有其他数据库回滚其操作，这样该全局事务

被回滚。

XA 规范对于应用来说，最大好处在于事务的完整性由交易中间件和数据库通过 XA 接口控制，应用程序只需要关注操作数据库的应用逻辑的处理，而无需过多关心事务的完整性，应用设计开发会简化很多。

3.3 IPC 机制简介

IPC 是 Inter-Process Communication，进程间通信。除了管道之外，Tuxedo 常用的 IPC 有 3 种：信号量、共享内存、消息队列。

3.3.1 命名管道

管道分为两种：管道和命名管道。

管道是 UNIX 系统 IPC 的最古老形式，并且所有的 UNIX 系统都提供这种通信机制。可以在有亲缘关系（父子进程或者是兄弟进程之间）进行通信，管道的数据只能单向流动，如果想双向流动，必须创建两个管道。

管道应用的一个重大缺陷就是没有名字，因此只能用于亲缘进程之间的通信。后来以管道为基础提出命名管道（named pipe，FIFO）的概念，该限制得到了克服。FIFO 不同于管道之处在于它提供一个路径名与之关联，以 FIFO 的文件形式存在于文件系统中。这样，即使与 FIFO 的创建进程不存在亲缘关系的进程，只要可以访问该路径，就能够彼此通过 FIFO 相互通信（能够访问该路径的进程以及 FIFO 的创建进程之间），因此，通过 FIFO 不相关的进程也能交换数据。值得注意的是，FIFO 严格遵循先进先出（first in first out）规则，对管道及 FIFO 的读总是从开始处返回数据，对它们的写则是把数据添加到末尾。它们不支持诸如 lseek()等文件定位操作。

3.3.2 消息队列

消息队列（message queue）是一个结构化的排序内存段表，该段表可以被多个进程共享，用来存放或者检索数据。每个消息队列都有一个队列头，用来描述消息队列的大量信息（队列键值、用户 ID、消息数目和最近读写消息队列的进程 ID），消息可以看作一个记录，具有特定的格式以及特定的优先级。对消息队列有写权限的进程可以按照一定的规则添加消息，对消息队列有读权限的进程可以从消息队列中读取消息。消息队列是随内核持续的，只有内核重启或者显示的删除一个消息队列时，该消息队列才会真正被删除。

Tuxedo 在客户机和服务器通信中大量使用 UNIX 系统的消息队列，消息包括客户请求、服务响应、会话消息、通知消息、管理消息、事物控制消息等。默认情况下每个服务器都有一个请求队列来接受客户端的请求，这就是 SSSQ（Single Server Single Queue）模式，每个客户机都有一个响应队列来接受服务器的响应消息，如果服务器太多，也可以配置多个服务器共享同一个请求队列，这就是 MSSQ（Multiple Servers Single Queue）模式。如果

一个服务器调用了其他服务，还需要为它创建一个响应队列。

3.3.3 信号量

信号量（semaphore）是为那些访问相同资源的进程以及同一进程不同线程之间提供的一个同步机制。它不是用于传输数据，而只是简单地协调对共享资源的访问。

信号量包含一个计数器，表示某个资源正在被访问和访问的次数，用来控制多进程对共享数据的访问。一旦成功拥有了一个信号量，对它所能做的操作只有两种：请求和释放。当执行释放操作时，系统将该信号值减 1（如果小于零，则设置为零）；当执行请求操作时，系统将该信号值加 1，如果加 1 后的值大于设定的最大值，那么系统将会挂起处理进程，直到信号值小于最大值为止。

Tuxedo 用信号量来确保在某一时刻只有一个进程对某一块共享内存进程访问。信号量配置太低会导致 Tuxedo 系统应用程序无法启动。

3.3.4 共享内存

共享内存（shared memory）是一片指定的物理内存区域，这个区域通常是在存放正常程序数据区域的外面，它允许两个或多个进程共享一给定的存储区，是针对其他通信机制运行效率较低而设计的。因为数据不需要来回复制，更不需要在客户机和服务器之间复制，进程可以直接读写内存，所以是最快的一种进程间通信机制。为了实现更安全通信，它往往与其他通信机制，如信号量结合使用，来达到进程间的同步及互斥。

在 Tuxedo 中，就是利用这个特性，使用共享内存存储公告牌，用来公告进程状态信息和需要在进程间共享或传递的数据，大大提高了对这些信息访问的效率。

3.3.5 IPC 资源相关的操作系统内核参数

由于 Tuxedo 大量采用 IPC 技术，实现“无连接”通信，提高系统性能，在部署 Tuxedo 应用时需要调整系统内核 IPC 参数，以达到良好运转。

更多关于 IPC 的详细内容，请参看后续第 17 章。

第 4 章　Tuxedo 的基本概念

Tuxedo 应用系统的部署可以划分为以下几个层次：域（Domain），逻辑机器（Machine），服务器组（Group），服务进程（Server），服务（Service）。Tuxedo 的配置文件，称为 UBBCONFIG 或 ubb，是对这些部署信息的定义。运行前，需要把 UBBCONFIG 装载成二进制文件，称为 TUXCONFIG。

4.1　域 Domain

4.1.1　域的概念和范围

Domain 概念：一个 Tuxedo 应用系统就是一个 Domain。

域范围：Tuxedo 的域特性把客户/服务器模型扩展到多个应用系统. 一个域既可以是一组 Tuxedo 的应用程序——若干相关的应用服务和配置环境的组合。域同时也可能是一组运行在另一个非 Tuxedo 环境中的应用程序。 Tuxedo 和 WebLogic 应用系统的互操作就是利用域的概念来实现的。

4.1.2　为什么要使用域

为了有效实现与其他系统的互连，Tuxedo 提出了 Domain（域）的概念，将由多台服务器共同组成的应用系统按功能或结构划分为不同的域，每个域独立地完成域内的操作，域间操作由域网关完成，从而提高每个域和整个系统的运行效率。不同的 Tuxedo 应用域中的服务程序可以互相访问对方的服务，并且当一个交易同时执行多个应用域中的服务(即对于分布式事务处理）时，能够确保交易的完整性。同时，Tuxedo 系统可以指定哪些服务是可供外部应用域访问的，并可为这些服务设置访问控制表等安全认证手段，提高整个系统的安全性。

Tuxedo 应用系统：一个 Tuxedo 应用系统是由在一个 TUXCONFIG 文件中定义的资源及其客户端的总称，它只能有一个 TUXCONFIG 文件，一个 Tuxedo 应用系统能够通过域网关与别的 Tuxedo 应用系统或其他的应用中间件系统实现互操作。

4.2　逻辑机器 Machine

4.2.1　Machine 的概念和范围

Tuxedo 对分布在多台机器上的同一个应用采用集中管理，因此 Machine 是在 Tuxedo

配置文件 UBBCONFIG 中第二个需要配置的段，也是必须配置的段。它主要涵盖了一台物理服务器的基本信息，如主机名、Tuxedo 安装路径、应用程序路径、Tuxedo 配置文件路径等。在一个 UBBCONFIG 配置文件中可以配置多个 Machine，即 MP 模式。

4.2.2 为什么使用 Machine

Machine 节点中定义了 Tuxedo 应用系统的基本信息，如相关文件的存放位置以及本机可提供的客户端最大访问量、可提供的最多服务数等基本信息。

4.3 服务器组 Group

4.3.1 组的概念和范围

Group 只是一个逻辑概念，并不真实存在，它下面是一个个的 Server，每个 Server 也就是一个进程，同时每个 Group 又属于一个 Machine。

4.3.2 为什么要使用组

Group 在逻辑上起到承上启下的作用。一个应用可以开启多个进程，每个进程可以属于不同的组，组又可以指定到不同的 Machine。因此可以通过增加物理机器 Machine 和组 Group 以及 Server 来增加系统的处理能力实现高负载。组同样也是消息路由以及服务迁移的单位。在 XA 事务环境中，组还与资源管理器相关联。

4.4 服务进程 Server 和服务 Service

4.4.1 什么是 Server 和 Service

Tuxedo Server 就是应用系统的一个进程，提供了对数据库或其他集中式资源的访问。

Tuxedo Service 从业务逻辑上来说就是对外提供的一种服务，是运行在 Server 上的一个函数，可命名的，客户端在调用时只需要知道 Service 的名字即可，不用知道 Service 运行在哪台机器的哪个 Server 上。

4.4.2 Server 和 Service 的关系

Server 是服务器上的一个进程，可以理解为一个程序，程序里有很多函数，而这些函数就是 Service。Service 需要注册到 Server 里才能被客户端调用。

4.4.3　服务进程中的主要函数

Tuxedo 服务程序启动后，它总是保持运行状态，直到接收到一个 shutdown 命令为止。一个典型的 Tuxedo 服务程序在 shutdown 或 reboot 之前都在执行着数千个服务。部分服务进程中的函数如下。

（1）在 Tuxedo 服务程序启动时，执行 tpsvrinit()函数，可以打开一些如数据库之类的资源供以后使用。

（2）在 Tuxedo 服务程序终止时，执行 tpsvrdown()函数，可以关闭 tpsvrinit()中打开的资源。

（3）Tuxedo 服务程序以服务的形式来响应客户程序的请求，客户程序不是直接指定调用的服务程序的，而是调用服务，客户程序不知道处理它请求的服务程序的位置。

（4）服务程序调用 tpreturn()函数来结束服务请求，并返回一个缓冲区，必要时，将它传给客户程序。

4.5　Tuxedo 通信方式综述

ATMI 环境支持的 C/S 通信方式有：请求/应答式通信、会话通信、队列通信、事件代理通信和消息通知。

4.5.1　请求/应答式通信

请求/应答式通信是发生在客户机和服务器之间的一种简单会话模式，客户机发出一个请求，服务器做出单个响应。

Tuxedo 使用 IPC（Inter-Process Communication）消息队列来实现请求/应答式通信。消息队列是实现面向无连接通信的关键技术，Tuxedo 系统会给每一个服务进程分配一个 IPC 消息队列，称为请求消息队列，每个客户机分配一个响应队列。这样，客户机和服务器之间就不需要建立通信连接，客户机把请求消息放入服务器的请求队列中，然后从自己的响应队列中检查响应结果。

请求/应答式通信如图 4-1 所示。

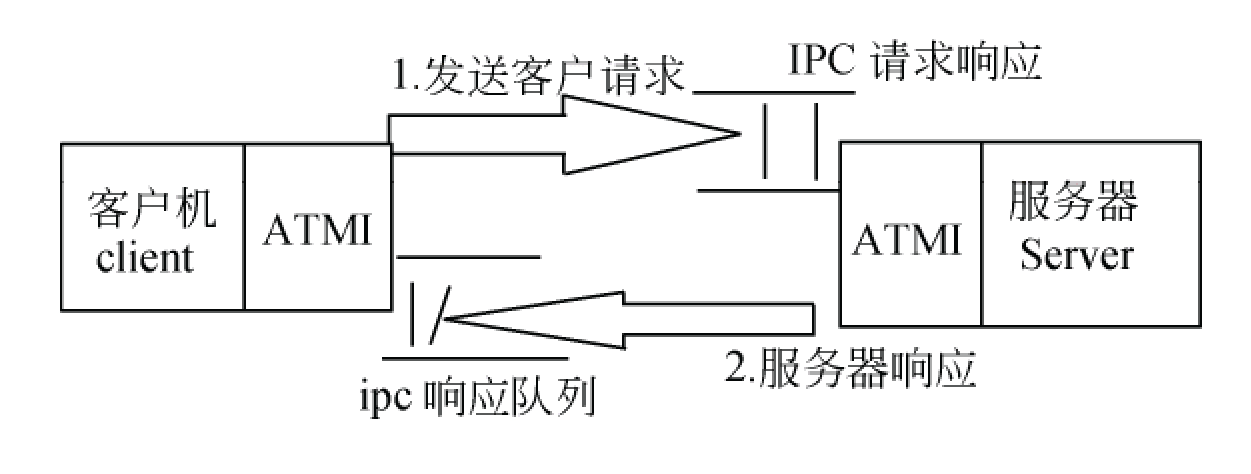


图 4-1

请求/应答式通信又可以分为同步调用、异步调用、嵌套调用和转发调用。

1．同步方式 tpcall()

如图 4-2 所示，客户程序使用 tpcall()调用把一个请求送到服务 X，然后就失去了控制权。X 服务处理完请求后，通过 tpreturn()调用将控制权返回给客户程序。客户程序得到控制权后，接着执行 tpcall()以后的流程。也就是说，tpcall()调用是阻塞的。

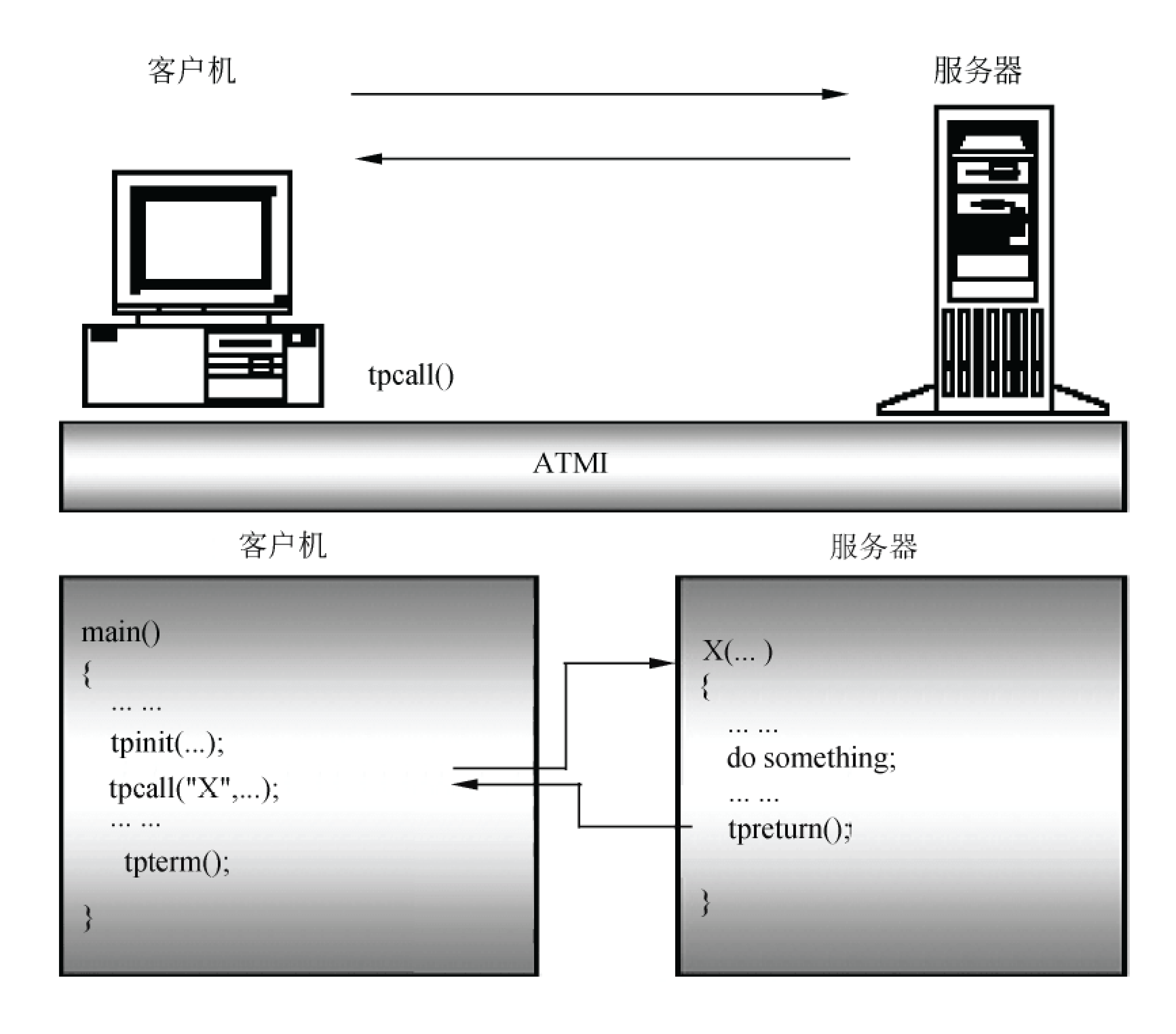


图 4-2

下面是一段同步通信的例程。

示例 4-1：

```
char *buf, *sendstr="Yesterday once more";
tpinit ( (TPINIT *)  NULL) ;
   buf = (char *) tpalloc ("STRING", NULL, len) ;
    (void) strcpy (buf, sendstr) ;
   tpcall ("STRING_SERV", buf, 0, &buf, &len, 0) ;
   printf ("Returned string is: %s\n", buf) ;
   tpfree (buf) ;
 tpterm () ;
```

2．异步方式 tpacall()

如图 4-3 所示，客户程序使用 tpacall()调用把一个请求送到服务 X，但并未失去控制权，仍接着执行后面的流程。X 服务处理完请求后，通过 tpreturn()调用返回。客户程序需要 X 服务的返回结果时，通过 tpgetrply()调用获得。

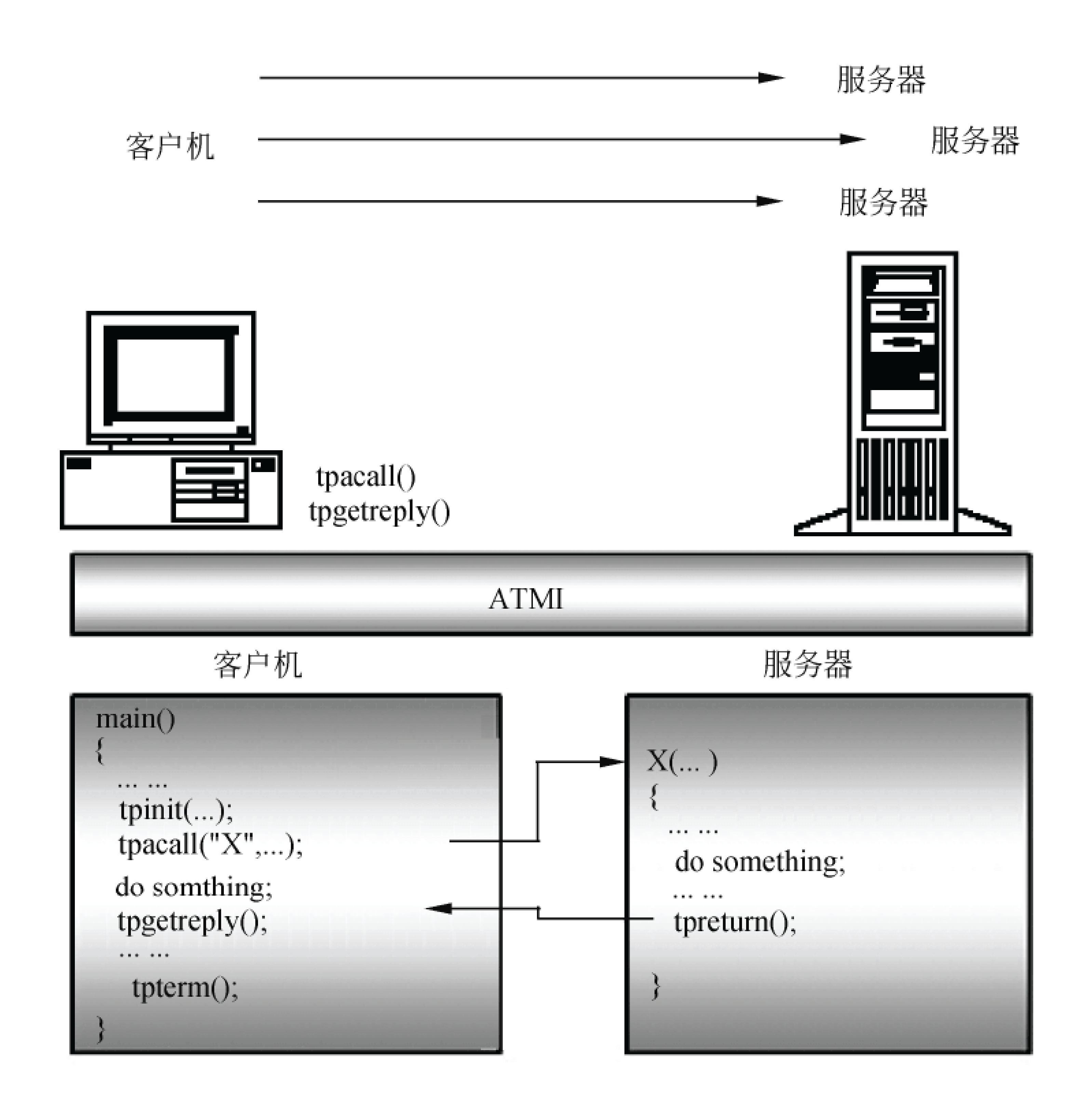


图 4-3

下面是一段异步通信的例程。

示例 4-2：

```
int cd;
tpinit ( (TPINIT *)  NULL) ;
   buf= (char *)  tpalloc ("STRING", NULL, len) ;
   cd=tpacall ("STRING SERV", buf, len, 0) ;
   tpgetrply (&cd, &buf, &len, 0) ;
   printf ("Returned string is: %s\n", buf) ;
   tpfree (buf) ;
tpterm () ;
```

3．嵌套调用

嵌套调用：一个 Tuxedo 服务可以充当客户程序，去调用另一个 Tuxedo 服务器中的服务。

如图 4-4 所示，客户程序调用了 Server1 中的 X 服务，而 X 服务又调用了 Server2 中的 Y 服务。Y 服务处理完毕后，使用 tpreturn()调用将控制权返回给 X 服务，X 服务再通过 tpreturn()调用将控制权返回给客户程序。嵌套调用具有效率高的特点，因为它可以充分利用现有的资源。

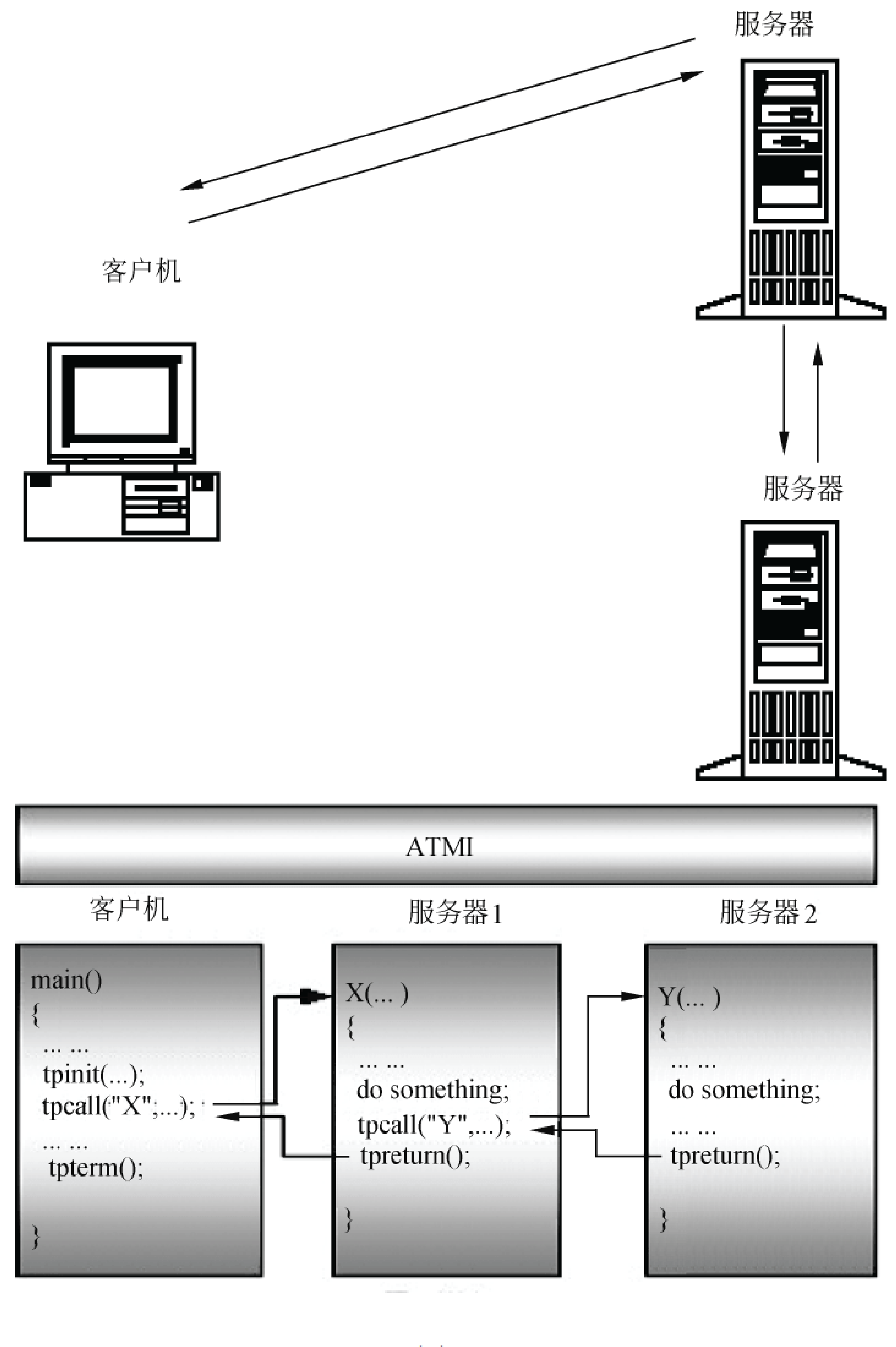


图 4-4

如银行的“转账”服务就可以通过嵌套调用“取款”和“存款”两个服务来完成。

示例 4-3：

```
TRANSFER (... ...)
{
  tpcall ("WITHDRAWAL",...) ;
  ... ...
  tpcall ("DEPOSIT",...) ;
  tpreturn () ;
};
```

4. 转发调用 tpforward()

转发调用和嵌套调用类似，所不同的是最里层嵌套服务可以直接给客户程序一个响

应，而不必按照调用栈逐级返回，中间服务只是简单地使用 tpforward()调用将客户请求转发给其他服务来处理。

如图 4-5 所示，Server1 上的 X 服务将请求转发给 Server2 上的 Y 服务来处理。Y 服务处理完毕后，通过 tpreturn()调用直接将结果返回给客户端。

在转发调用中，第一个服务是一个分发代理，它可以对其他服务器的负载状况进行判断，将请求路由到负载较轻的服务器，这样就可以达到均衡负载的效果。

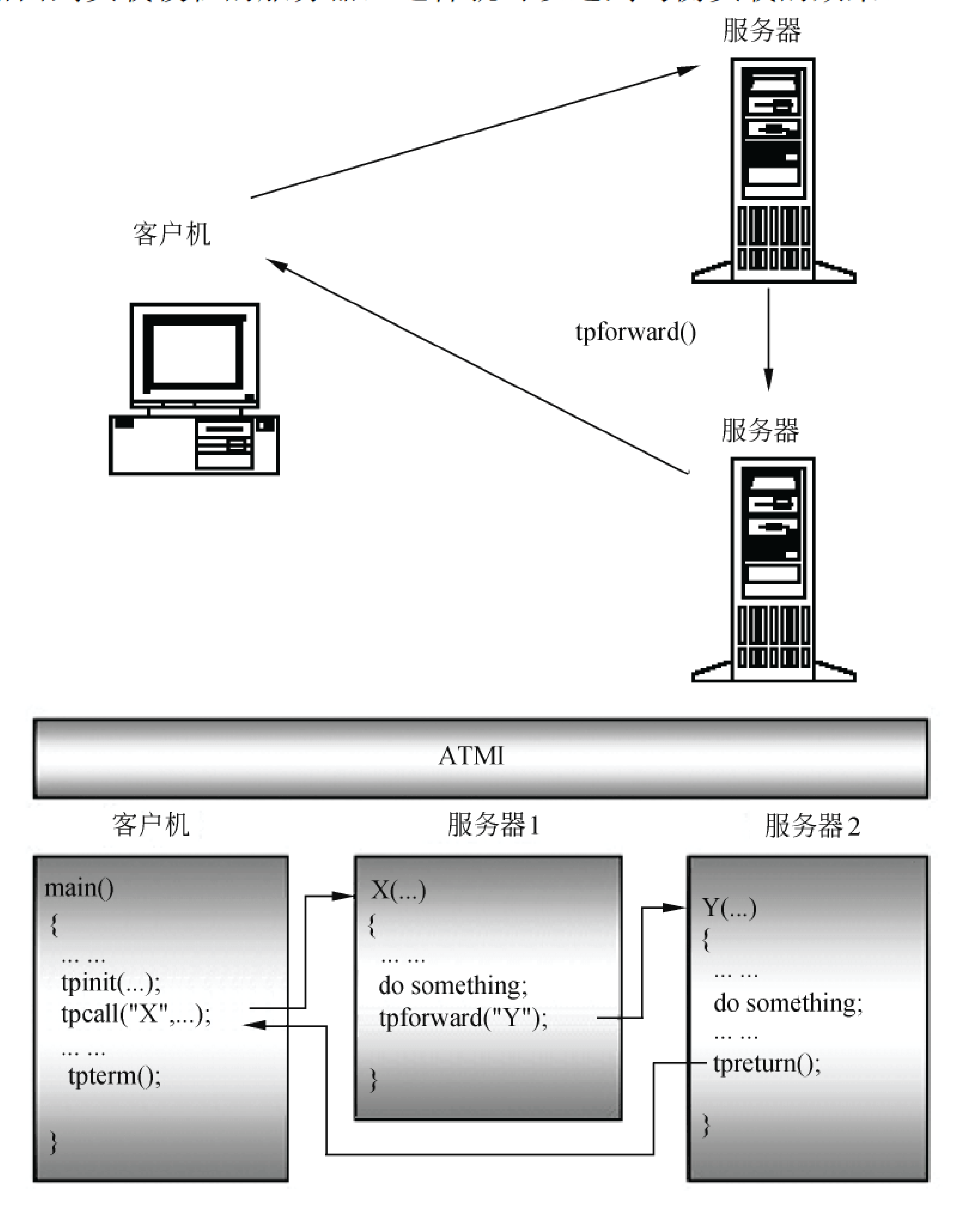


图 4-5

4.5.2　会话方式 tpsend()/tprecv()

会话通信适用于有多个缓冲区需要以有状态的方式在 Tuxedo 客户程序和服务器之间传递的场合。

如图 4-6 所示，客户程序通过 tpconnect()调用接连到 X 服务，双方通过 tpsend()和 tprecv()

进行若干个来回的会话后，X 服务通过 tpreturn()或 tpdiscon()调用来退出会话。

使用会话通信时要非常小心，若一方结束会话而未通知另外一方，则另外一方就一直处于会话状态。

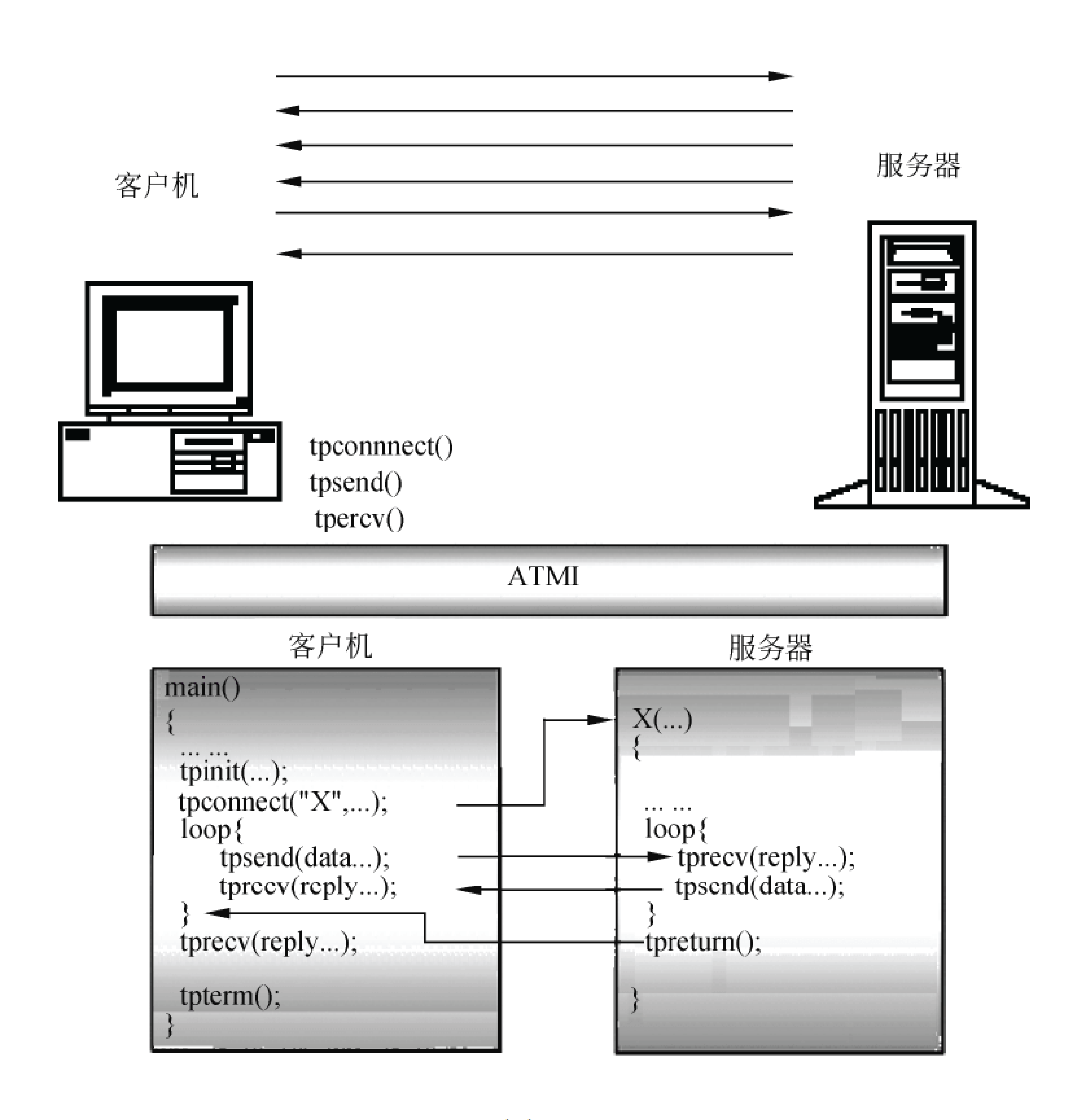


图 4-6

在下面的程序段中，左侧客户程序建立一个和服务 Y_SERV 的会话连接，然后向它发送字符串，Y_SERV 调用 STRING_SERV 服务将字符串转换成大写后，返回客户端，当收到"quit"串时，双方都退出循环，结束会话。

示例 4-4：

```
main (int argc, char *argv[]) {
……
   tpinit ( (TPINIT *)  NULL) ;
   line =  (char *)  tpalloc ("STRING", NULL, len) ;
   cd = tpconnect ("Y_SERV", line,0,TPSENDONLY) ;
   while (1) {
      (void) gets (reply) ;
      (void)  strcpy (line,reply) ;
     tpsend (cd,line,0,TPRECVONLY,&revent) ;
```

```
        if (!strcmp (reply,"quit") ) break;
        tprecv (cd,&line,&len, TPNOCHANGE,&revent) ;
        printf (":%s\n", line) ;
      }
   }

   Y_SERV (TPSVCINFO *svcinfo) {
   ……
      line = tpalloc ("STRING",NULL,80+1) ;
      while (1) {
         tprecv (svcinfo->cd,&line,&len,TPNOCHANGE,&len) ;
         if (strcmp (line,"quit") ==0) {
            tpreturn (TPSUCCESS,0,NULL,0,0) ;
            break;
         }
         else{
            tpcall ("STRING_SERV",line,0,&line,&len,0) ;
            tpsend (svcinfo->cd,line,0, TPRECVONLY,&len) ;
         }
      }
   }
```

对会话服务器需要在配置文件的*SERVER 段中加以说明，比如：

```
   Y_SERV SRVGRP=GROUP1 SRVID=2 CONV=Y MIN=1 MAX=10 RQADDR="y_serv"
```

它表示 Y_SERV 是会话服务器（CONV=Y），它最多同时可启动 10 个会话，请求队列地址是 y_serv。

4.5.3　通知广播 tpnotify()/tpbroadcast()

Tuxedo 系统提供了一种基于事件的通信机制，通过这种机制，系统进程之间可以相互转递通知消息。Tuxedo 支持两种类型的事件通信：通告和代理。通告（也称为广播）是指消息发送方直接将消息发送给接收方，而代理是指通信双方通过一个匿名的事件代理中介来完成消息的接收和发送。通告消息可以使用 tpnotify()或 tpbroadcast()调用来发送，前者只将事件通知给单个通信实体，后者可以将事件广播给一个或多个通信实体。

图 4-7 展示了一个消息通告的例子。X 服务使用 tpnotify()调用给客户机发送通知消息，客户机使用 tpsetunsol()调用来设置一个消息处理器 func()。X 服务给客户机发送通知消息时，并不会影响客户机和服务器正在进行的“请求/应答”或会话通信。在一个非信号系统中，客户机必须使用 tpchkunsol()调用来检查意外事件，而在信号系统中，tpchkunsol()什么也不做，立即返回。

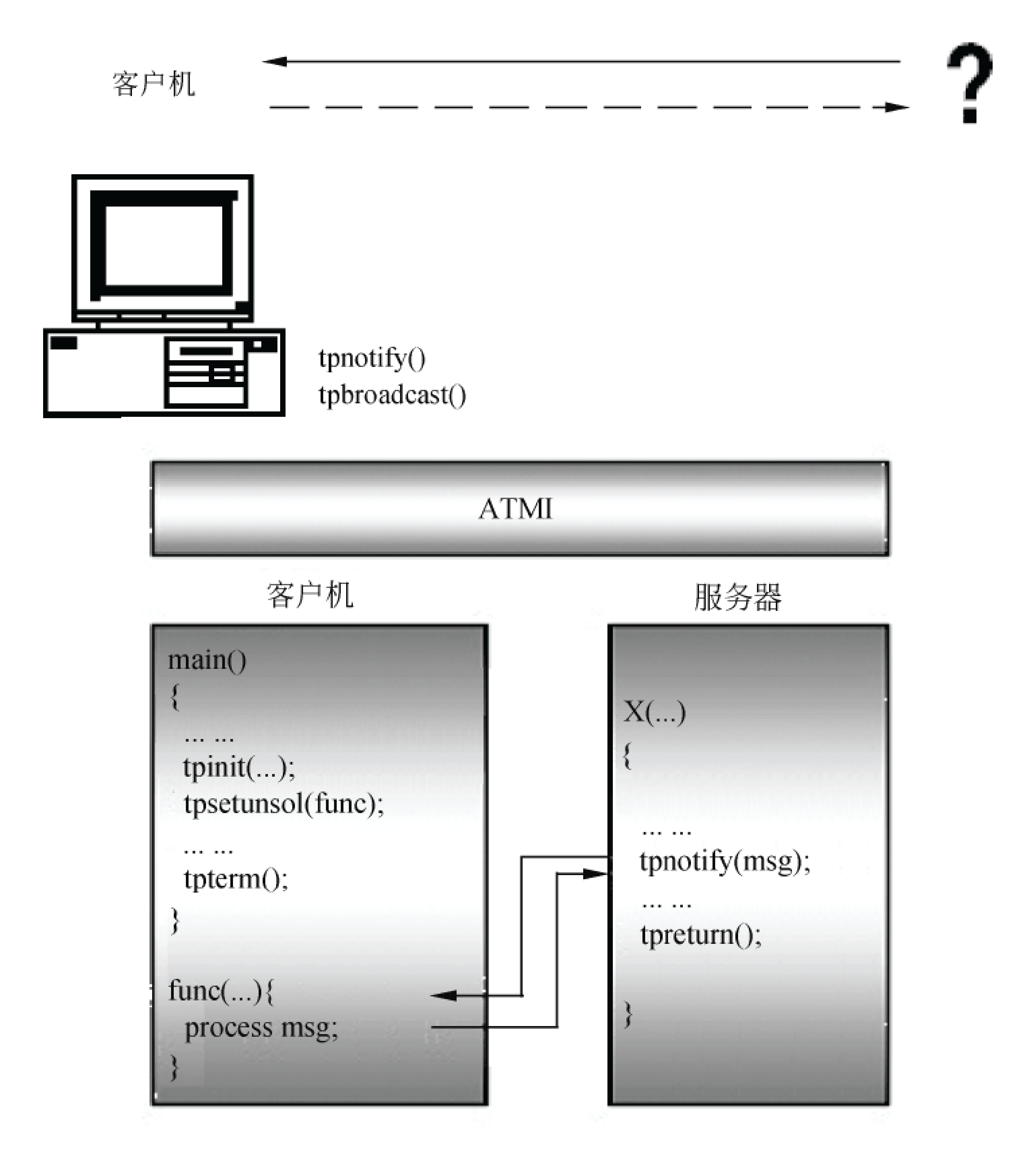


图 4-7

下面的代码段演示了事件通告的处理过程。

示例 4-5：

```
void func (char *, long, long) ;
int main (int argc, char *argv[]) {
   TPINIT * tpinfop;
   void  (**p)  () ;
   long len;
   p=&func;
   tpinfop = tpalloc ("TPINIT", NULL,TPINITNEED (0) ) ;
   strcpy (tpinfop→usrname, " landingbj ") ;
   strcpy (tpinfop→cltname, "landingbj") ;
   tpinfop→flags = TPU_SIG;
   tpinit (tpinfop) ;
   tpsetunsol (p) ;
   tpcall ("X",NULL, 0, &line, &len, 0) ;
   tpterm () ;
}
```

```
void func (char * string, long len, long flag) {
  printf ("%s\n",string) ;
  return;
}

X (TPSVCINFO *rqst)
{
  char * line;
  line = tpalloc ("STRING", NULL, 80+1) ;
  strcpy (line,"notification from X service") ;
  tpnotify ( (CLIENTID *) &rqst→cltid, line, 0L, TPNOBLOCK) ;
  //tpbroadcast (NULL," landingbj "," landingbj ",line , 0, TPSIGRSTRT) ;
  tpreturn (TPSUCCESS, 0, rqst→data, 0L, 0) ;
}
```

上面的部分是客户程序，它定义了一个事件处理器 func，func 的声明必须遵循如下格式。

```
void func (char *, long, long) ;
```

第一个参数定义了事件的消息体，它是一个类型缓冲区；第二个参数定义了消息体的长度，第三个参数是标志位。

void （**p）()定义了一个指向函数指针的指针 p，它指向事件处理器 func。tpalloc()调用申请了一块 TPINIT 类型的缓冲区 tpinfop，tpinfop 中填入了用户名、客户机名和信号标记 TPU_SIG。tpsetunsol（p）调用将 p 指向的函数 func 设置为客户机的事件处理器。

下面的部分是服务程序，它通过 tpnotify()调用向 rqst→cltid 客户机发送通知消息，当然也可以使用 tpbroadcast()调用来给 jq 客户机发送消息。若需要给 SITE1 上所有的客户机发送通知消息，可以用通过如下语句来完成。

```
tpbroadcast ("SITE1",NULL,NULL,...) ;
```

客户程序对通知消息的获取模式可以通过配置文件中*RESOURCE 段的 NOTIFY 参数来设置，NOTIFY 可取 SIGNAL、DIPIN、IGNORE 3 个值，分别对应 tpinfop→flags 的 TPU_SIG、TPU_DIP 和 TPU_IGN 3 个值，但如果显式地指定了 tpinfop→flags，则 NOTIFY 参数将被忽略。

4.5.4　事件代理 tppost()/tpsubscribe()

事件代理也就是消息发布/订阅，在这种模型中，客户机和服务器可以随时订阅和发布消息。Tuxedo 提供了两个事件代理器（TMUSREVT 和 TMSYSEVT）来处理订阅请求，维护订阅事件列表和触发订阅者指定的操作，如图 4-8 所示。

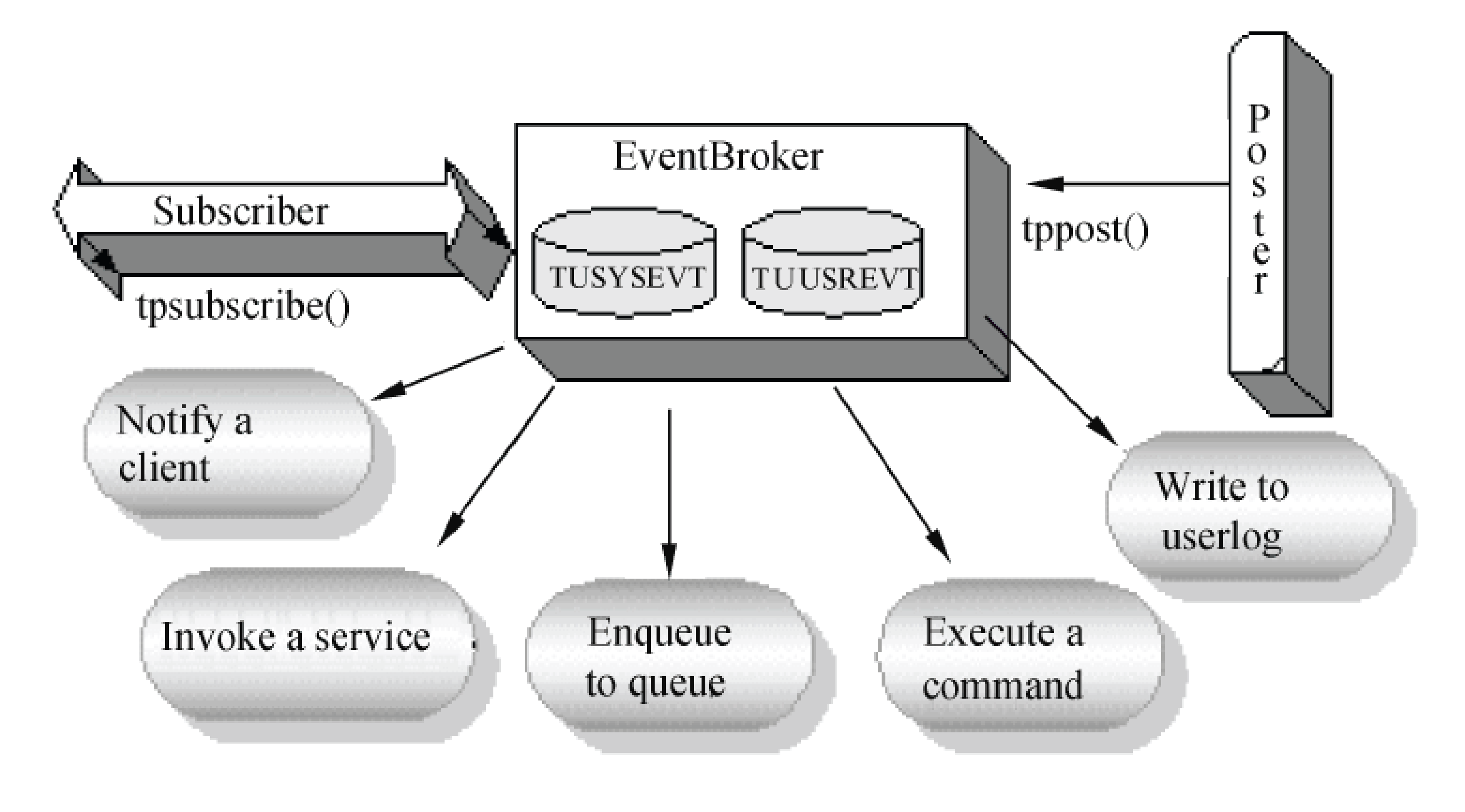


图 4-8

当客户机订阅的事件被触发时，事件代理服务器根据订阅要求执行以下操作。

（1）给客户端发一个通知消息。

（2）执行一个服务调用。

（3）往某个消息队列中放入一条消息。

（4）写一条 ULOG 日志。

（5）执行一个命令脚本。

4.5.5 队列存储 tpenqueue()/tpdequeue()

Tuxedo /Q 提供了一种队列机制，它允许消息按照某种排队规则（如 fifo，filo 等）暂时存储到介质中，等待其他进程对其进行处理。Tuxedo/Q 用到了 Tuxedo 提供的两个服务器：消息队列服务器 TMQUEUE 和消息转发服务器 TMQFORWARD。TMQUEUE 用于对消息进行出队入队管理，TMQFORWARD 用于将消息从队列中取出，转发给其他服务器进行处理，并将处理结果放回响应队列。

Tuxedo 的队列通信有两种类型：客户对客户和客户对服务器。

对于前者，通信双方是对等的，彼此只需通过 tpenqueue()调用将消息放入对方的接收队列，通过 tpdequeue()调用从自己的队列中取出消息，不涉及到消息转发，因此只用到了 Tuxedo 的 TMQUEUE 服务器，如图 4-9 所示。

对于后者，可以通过转发服务器 TMQFORWARD 将客户请求转发给相应的服务进行处理，如图 4-10 所示。

下面介绍一种通过管道通信自动创建队列空间和队列的方法。qmadmin 是 Tuxedo 提供的一个用于创建队列的实用程序，它的操作方式是交互式的，如果完全靠手工输入信息不是很方便，而且容易出错，一旦有一个操作失误，就得重头开始，因此不适合于创建大量的队列。但幸运的是 Tuxedo 的实用程序都支持管道通信，所以只要建立一个与 qmadmin 的通信管道，把指令按顺序输入管道，就可以达到自动创建队列的效果。

请看下面的程序段。

示例 4-6：

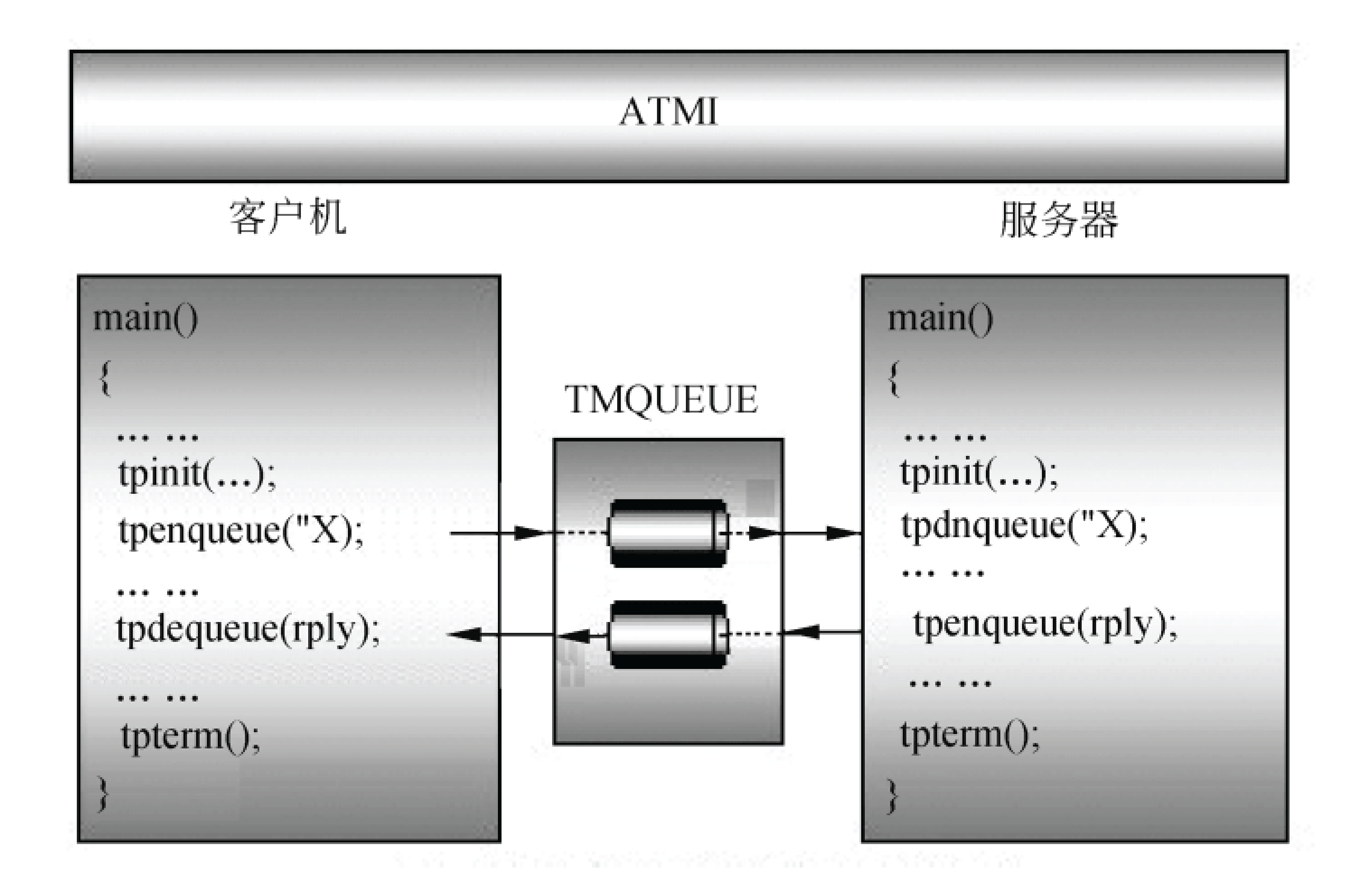


图 4-9

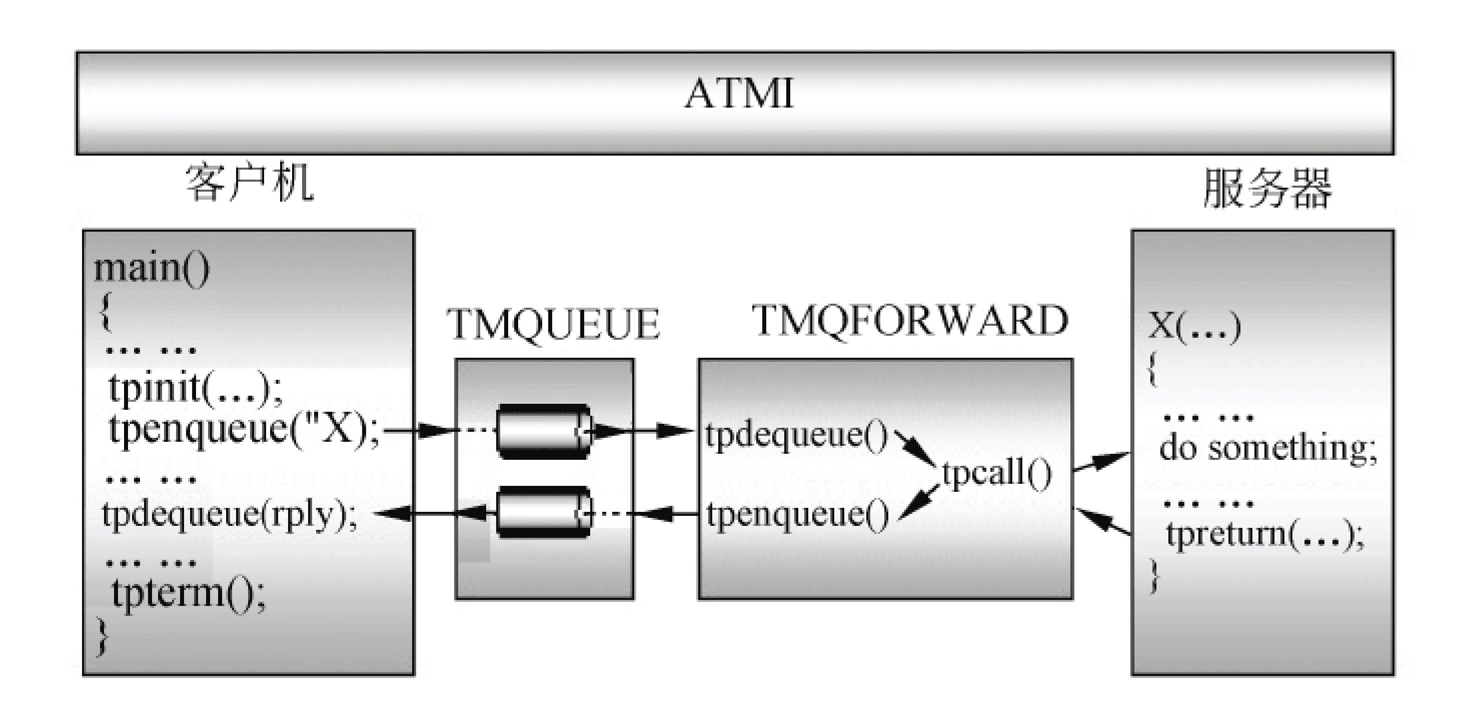


图 4-10

```
fd =  popen ("qmadmin > nul", "wt") ;
fprintf (fd, "crdl %s 0 500\n", getenv ("QMCONFIG") ) ;
fprintf (fd, "qspacecreate QSP_BANKAPP 88946 200 20 5 5 200
Q_ERROR 'y' 16\n") ;
fprintf (fd, "qopen QSP_BANKAPP\n") ;
fprintf (fd, "qcreate Q_ERROR fifo none 0 0 100%% 0 \"\"\n") ;
fprintf (fd, "qcreate Q_OPENACCT_LOG fifo none 2 2 5m 0m
\"tmboot -s TMQFORWARD\"\n") ;
fprintf (fd, "qcreate Q OPENACCT fifo none 0 0 100%% 0%% \"\"\n") ;
fprintf (fd, "qclose\n") ;
exit ( _pclose (fd) ) ;
```

它使用_popen 函数创建一个通向 qmadmin 的管道，然后通过 crdl 指令创建了环境变量 QMCONFIG 指定的队列设备，通过 qspacecreate 创建了一个名为 QSP_BANKAPP 的队列空间，通过 qcreate 创建了 3 个队列：Q_ERROR、Q_OPENACCT_LOG、Q_OPENACCT。

4.6 Tuxedo 多机部署

4.6.1 Tuxedo 集群

Tuxedo 集群系统中，一个域中只有一个配置文件，即 TUXCONFIG 存放在 Master 机器上，在执行 tmadmin 管理命令进行管理时也在 Master 机器上运行，其他非 Master 机器只负责干活并没有管理权限。多系统多机之间实现通信需要每台机器上都有一个 Bridge 进程，然后通过 TCP/IP 协议进行通信。Bridge 进程维持一个长连接，一旦建立不会断掉。

在 Master 机器上执行启动命令，会根据 TUXCONFIG 配置文件中的配置信息启动配置的非 Master 机器。Master 机器主动去和非 Master 机器建立连接，此时就需要在非 Master 机器上有一个守候进程，在某个端口进行监听也就是 listen 进程。

tlisten 进程需要手动启动，一般 Master 机器上也需要启动 tlisten 进程。

4.6.2 多套 Tuxedo 应用之间的通信

一个 TUXCONFIG 配置了一套 Tuxedo 应用。当多套 Tuxedo 应用之间需要互相调用服务，或进行全局事务处理时，可以将它们通过 Tuxedo 域网关连接起来。

Tuxedo 中每个域是一个独立自治的系统，各个 Tuxedo 域之间可以定义不同的连接策略，进行连接控制，互相发布服务，控制访问权限，支持跨 Tuxedo 域的全局事务等。详细配置过程请参见 Tuxedo 域配置文件。

4.7 Tuxedo 远程客户端

4.7.1 什么是远程客户端

Tuxedo 有两种客户端：本地客户端和远程客户端。本地客户端是指与 Tuxedo 服务器在同一台机器上，不用通过网络就能访问 Tuxedo 服务器的客户端。远程客户端是指通过网络才可以访问到 Tuxedo 服务器的客户端。

本地客户端与远程客户端的主要区别如下。

（1）本地客户端只能用 C 语言或 COBOL 语言编写，远程客户端可以用几乎所有的编程语言编写。

（2）在远程客户端所在的机器上需要安装 Tuxedo 客户端软件，并且要设置相应的环境变量，本地客户端不用做这些设置。

（3）用 buildclient 编译远程客户端要加-w，编译本地客户端则不用。

4.7.2　WSL/WSH 配置与工作机理

（1）在*MACHINES 段中要配置 MAXWSCLIENTS，即最多可以有多少个远程客户端同时连接到该服务器上。

（2）　在*SERVERS 段中要配置 SERVER：WSL。

WSL SRVGRP="GROUP1" SRVID=1116 CLOPT="-A -- -n //192.168.120.113:8888 -m 2 -M 5 -x 6"

说明：−n //192.168.120.113:8888 表示远程客户端通过该端口与服务器建立连接，-m 2 表示最少启动多少个 WSH 进程，-M 5 表示最多启动多少个 WSH 进程，-x 6 表示每个 WSH 进程可同时处理多少个远程客户端。

（3）　客户端所在的服务器上要配置 WSNADDR 环境变量，它的值为-n 参数的值，如配置为：　SET WSNADDR= //192.168.120.113:8888。

图 4-11 为 Tuxedo 应用系统客户端访问 Tuxedo 服务器上服务的过程图。

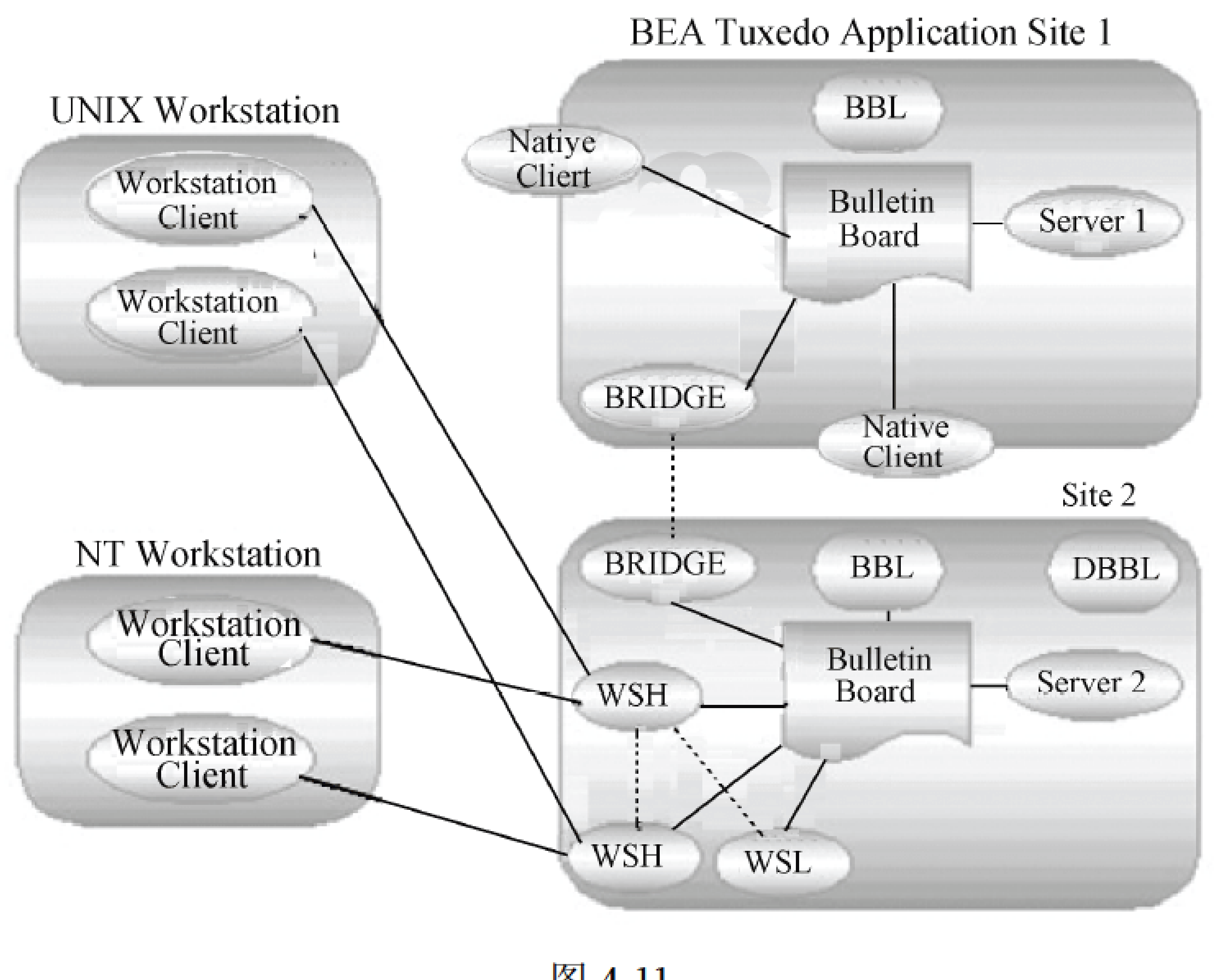


图 4-11

说明：

- **WSC（Workstation Client）**　用于指 Tuxedo 的客户端部分;
- **WSL（Workstation Listener）**　表示 Tuxedo 系统自带的一个 Server，它监听一个指定的端口，WSC 最初与该 SERVER 建立连接;
- **WSH（Workstation Handler）**　表示 Tuxedo 系统自带的一个 Server，由它处理 WSC 与 Tuxedo Server 之间的通信;
- **Bulletin Board（公告板）**　表示 Tuxedo 把系统的配置保存在一个共享内存中，该共享内存成为公告板（BB）;
- **BBL**　表示 Tuxedo 管理进程，主要对公告板等进行管理。

Workstation Client 与 Tuxedo Server 建立连接的过程为如下。

（1）WSC 调用 tpinit()或 tpchkauth()。

（2）WSC 采用在 WSNADDR 中指定的 IP 地址与服务器端的 WSL 建立连接。

（3）WSL 为该 WSC 指定一个 WSH，并把该 WSH 的监听端口返回给 WSC。

（4）WSC 采用返回的端口与指定的 WSH 建立连接，并与 WSL 断开连接，之后 WSC 与 Tuxedo Server 之间的通信通过 WSH 进行处理，与 WSL 无关。

（5）tpinit()或 tpchkauth()调用返回。

4.7.3 Java 远程客户端接入 Jolt

外部应用访问 Tuxedo 服务的情况很常见，一般有 WTC 和 Jolt 两种方法，Jolt 是 Tuxedo 自带的 jar 组件，在 Tuxedo 的安装过程中可以看到安装的 Jolt 组件。调用服务的步骤如下。

（1）先准备 Tuxedo 服务端代码。

（2）在 Tuxedo 中配置 Jolt 相关文件。

（3）启动 Tuxedo 服务。

（4）配置 WebLogic 服务与 Tuxedo Jolt 相关的参数。

（5）启动 WebLogic 服务。

（6）编写 Servlet 代码，运行调用服务。

开发建立一个 Jolt 客户端访问 Tuxedo 应用服务主要过程如下图 4-12 所示。

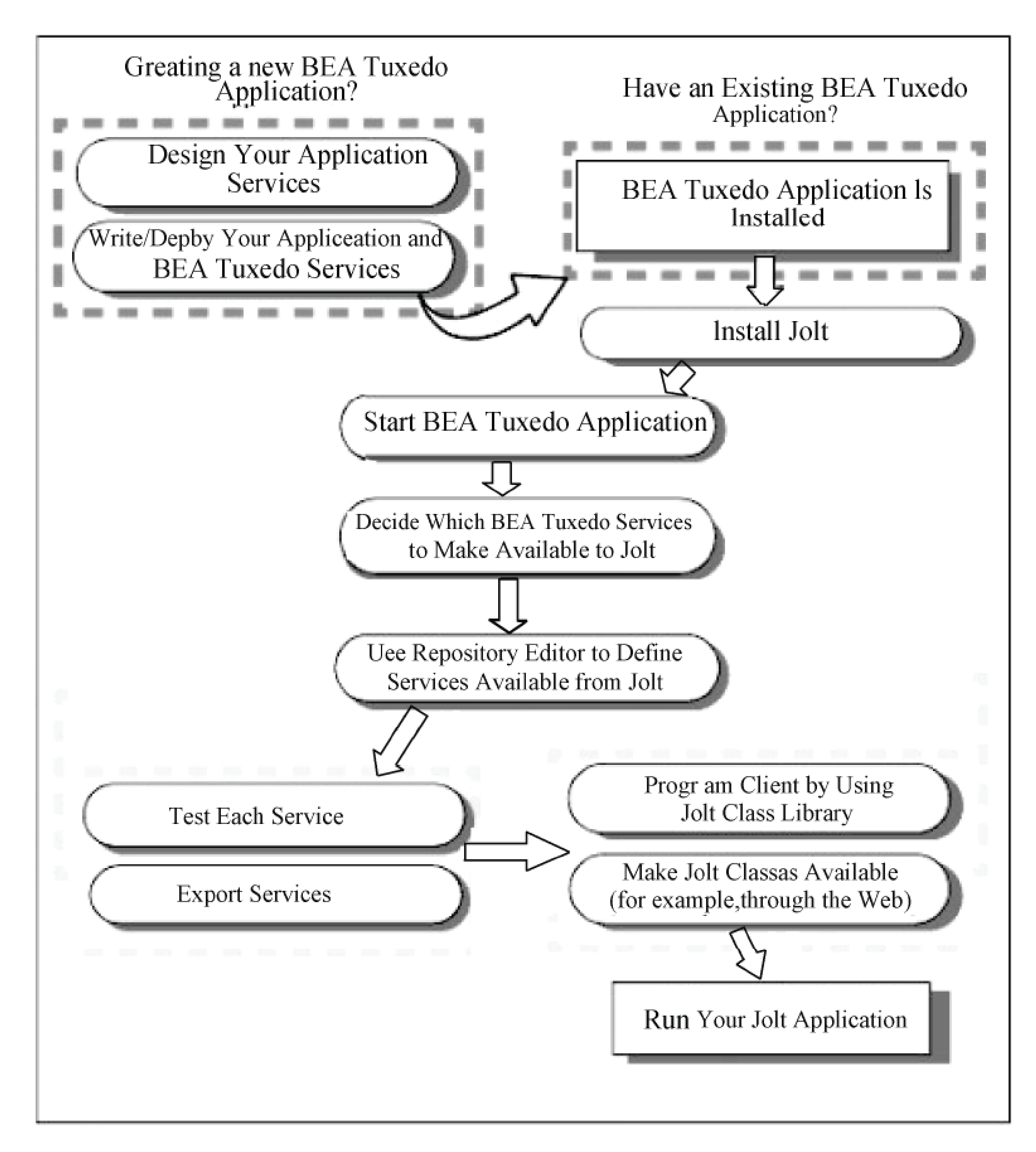


图 4-12

配置文件示例如下。

示例 4-7：

```
*RESOURCES
IPCKEY  123456
DOMAINID simpapp
MASTER  landingbj
MAXACCESSERS 150
MAXSERVERS 100
MAXSERVICES 100
MODEL  SHM
LDBAL  N

*MACHINES
LANDINGBJ  LMID = landingbj
TUXDIR = "/home/bea/tuxedo"
TUXCONFIG = "/home/bea/tuxedo/landingbj /tuxconfig"
APPDIR = "/home/tuxedo/landingbj "
MAXWSCLIENTS=1
TLOGDEVICE = "/home/tuxedo/landingbj /TLOG"
TLOGNAME=TLOG
TLOGSIZE = 100
*GROUPS
APPGRP LMID= landingbj GRPNO = 1
#OPENINFO="Oracle XA:Oracle XA+Acc=P/ landingbj / landingbj +SqlNet=Linux+
SesTm=600
+MaxCur=5+LogDir=."
#TMSNAME="TMS ORA9i" TMSCOUNT=2
JSLGRP  LMID=landingbj GRPNO = 2
JREPGRP LMID=landingbj GRPNO = 3
*SERVERS
server SRVGRP=APPGRP SRVID=1
#WSL SRVGRP=APPGRP SRVID =300
#CLOPT="-A -- -n //192.168.0.166:8888 -d/dev/tcp -m1 -M5 -x 10"
JSL SRVGRP=JSLGRP SRVID=301
CLOPT="-A -- -n //192.168.0.166:9878 -M 10 -x 10 -m 2"
JREPSVR SRVGRP=JREPGRP SRVID=302
CLOPT="-A -- -W -P /home/bea/tuxedo/udataobj/jolt/repository/jrepository"
*SERVICES
TOUPPER
```

该 config 文件是在*GROUPS 和*SERVERS 中各加了两项对应内容：

```
JSLGRP LMID=landingbj GRPNO = 2
JREPGRP LMID=landingbj GRPNO = 3
```

和

```
JSL SRVGRP=JSLGRP SRVID=301  CLOPT="-A -- -n //192.168.0.166:9878
-M 10 -x 10 -m 2"
JREPSVR SRVGRP=JREPGRP SRVID=302  CLOPT="-A -- -W -P /home/bea/tuxedo/
udataobj/jolt/repository/jrepository"
```

这两项是用于 Jolt 的访问接口。外部调用通过 JSL 和 JREPSVR 直接与 Tuxedo 服务通信。JSL 的主机地址是 Tuxedo 服务器地址，端口是随意指定的，不要与其他重复即可。在 config 里加入此内容后，config 文件就可用了。

配置 jrepository 文件，直接打开/home/bea/tuxedo/udataobj/jolt/repository/jrepository 文件添加以下内容：

```
add SVC/TOUPPER:vs=1:ex=1:bt=STRING:\ bp:pn=STRING
:pt=string:pf=167772161:pa=rw:ep:
add PKG/SIMPSERV: TOUPPER:
```

这个文件如果没有 TOUPPER 配置，调用会提示 TOUPPER not avaliable，TOUPPER 配置错误，会提示 TOUPPER has been modifed ...

对这个文件操作还有以下两种方式。

方式一：在 IE 中打开 E:\bea\tuxedo9.0\udataobj\jolt\RE.html，得到 Java applet，里面是图形界面，添加完服务，设置好参数即可。

方式二：在命令行键入 Java bea.jolt.admin.jbld //192.168.0.166:9878 services.rep，其中//192..要与 JSL 的设置一致，如下：

```
service=TOUPPER
inbuf=STRING
outbuf=STRING
export=true
param=STRING
type=string
access=inout
```

要执行这个命令设 JDK 环境变量是必不可少的，CLASSPATH 加入了以下内容：

```
/home/bea/tuxedo/udataobj/jolt/joltwls.jar;
/home/bea/tuxedo/udataobj/jolt/joltjse.jar;
/home/bea/tuxedo/udataobj/jolt/jolt.jar;
/home/bea/tuxedo/udataobj/jolt/joltadmin.jar
```

第 5 章　Tuxedo 主要的目录结构

5.1　总体目录结构分布

Tuxedo 总体目录结构：

```
|-bea                               bea 的主目录
    |-/ tuxedo                      Tuxedo 的总目录
      |-tux.env                     主要是用来在 UNIX 系统中设置环境变量
      |-/bin                        Tuxedo 的所有命令和可执行文件的安装目录
      |-/cobinclude                 COBOL 应用需要的头文件
      |-/include                    C 或者是 C++的头文件目录
      |-/jre                        Java 运行环境（jre），提供 Java 虚拟机（JVM）
      |-/lib                        Tuxedo 的动态库文件目录
      |-/locale                     本地语言包
      |-/help                       帮助目录

      |-/samples                    Tuxedo 自带的几种应用例子
            |-/atmi                 ATMI 的应用例子，包括 simpapp
            |-/corba                CORBA 的应用例子
            |-/jolt                 jolt 的使用例子
      |-/udataobj                   Tuxedo 系统目录，其中有不同种类的目录和文件
      |                             Tuxedo 的 lic.txt 和 RM 也放在这个目录中

|-/uninstaller                      卸载 tuxedo 的脚本目录
```

5.2　可执行文件说明

执行目录 bin：

|-/bin

可执行文件主要用来支撑 Tuxedo 系统运行，以及管理和监控 Tuxedo 系统的运行情况，主要包括以下内容。

（1）系统 SERVER，如 BBL、BRIDGE、GWTDOMAIN。

（2）Tuxedo 的管理命令，如 tmadmin、tmboot。

5.3　系统目录 udataobj 提要

系统目录 udataobj：

|-/udataobj	
|-/security	包含默认的 LDAP 过滤文件（bea_ldap_filter.dat）
|	和 LLE 以及 SS 相关的文件
|-/jolt	Tuxedo 的 Jolt 组件
|-/java	运行 Java class 类的存档文件
|-/snmp	包含一个 etc 目录，其主要包含了 SNMP 代理
|-/etc	配置文件、MIB 文件和其他一些不同种类文件
|-/webgui	包含 Tuxedo 的管理控制台的 Java 和图像文件
|-lic.txt	包含 Tuxedo 产品的使用许可 License
|-tlisten.pw	包含在安装时写入的 tlisten 的密码

5.4　C 语言头文件和库

与 C 语言有关的头文件和库文件：

|-/lib	静态和动态文件库（Tuxedo 系统本身的函数库）
|-/include	包含 C 或 C++语言的头文件，如 OMG IDL 文件

第 6 章　Tuxedo 配置相关文件

6.1　几个关键环境变量

Tuxedo 系统在建立、启动和运行的过程中，要从环境变量中读取一些配置参数。下面用表格的形式对变量的含义进行说明，见表 6-1。

表 6-1

环境变量	含义和用法
TUXDIR	Tuxedo 系统的安装路径，只要是设置 TUXDIR，Tuxedo 的 build 系统命令就会自动到$TUXDIR/include 和 lib 目录下去找到头文件和库文件 例如：TUXDIR=/root/bea/tux; export TUXDIR
APPDIR	应用程序的存放路径。Tuxedo 启动 SERVER 时，会在$APPDIR 下查找执行文件 例如：APPDIR=/root/simpapp; export APPDIR
TUXCONFIG	应用程序的二进制配置文件名 例如：TUXCONFIG=$APPDIR/tuxconfig; export TUXCONFIG
PATH	要包含 Tuxedo 的 bin 目录 例如：PATH=$TUXDIR/bin:$JAVA_HOME/bin:$PATH; export PATH
INCLUDE	仅用于 NT 平台，用于指定 C 语言头文件
CC、CFLAGS	指定 C 编译器和编译选项，如果系统没有使用操作系统的编译器（如 gcc）就需要指定 CC 变量 例如：CC =gcc; export CC
LD_LIBRARY_PATH SHLIB_PATH LIBPATH、LIB	这几个变量的作用是一样的，需要包含$TUXDIR/lib 目录及其他公共库路径 LD_LIBRARY_PATH 用于一般 UNIX 系统 SHLIB_PATH 用于 HP—UNIX 系统 LIBPATH 用于 AIX 系统 LIB 用于 NT 平台

在 UNIX 平台下安装 Tuxedo 系统后，可以在安装目录中找到一个 tux.env 的脚本，执行这个脚本就可以完成一些最常用的环境变量设置。

Tuxedo 客户端环境变量见表 6-2。

表 6-2

环境变量	含义和用法
TUXCONFIG	本地客户机加入 Tuxedo 系统时，要从这个配置文件中查找公告板信息，仅对本地客户机有意义

续表

环境变量	含义和用法
WSNADDR	远程客户机要加入 Tuxedo 系统时，要从当地的 WSNADDR 环境变量中读取服务器上 WSL 进程监听的 IP 和端口，仅对远程客户端有意义
WSTYPE	一个字符串，用于指定/ws 的机器类型，仅对远程客户端有意义 当 WSTYPE 和服务器 TYPE 一致时，将不对远程客户端和服务器之间传送的数据进行编码/解码，以提高性能

6.2 系统配置文件 UBB 及其内容

配置文件 UBB 描述了 Tuxedo 应用程序的运行环境和部署方式。

下面是一个简单的 UBB 的配置模版。

示例 6-1：

```
#       (c) 2003 BEA Systems, Inc. All Rights Reserved.
#ident  "@(#) samples/atmi/simpapp/ubbsimple    $Revision: 1.7 $"

#Skeleton UBBCONFIG file for the Tuxedo Simple Application.
#Replace the <bracketed> items with the appropriate values.

*RESOURCES
#IPCKEY         <Replace with a valid IPC Key>

#Example:
IPCKEY          123456

DOMAINID        simpapp
MASTER          simple
MAXACCESSERS    10
MAXSERVERS      5
MAXSERVICES     10
MODEL           SHM
LDBAL           N

*MACHINES
DEFAULT:
                APPDIR="/home/landingbj/simpapp"
                TUXCONFIG="/home/landingbj/simpapp/tuxconfig"
                TUXDIR="/home/bea/tux"
#Example:
#               APPDIR="/home/me/simpapp"
#               TUXCONFIG="/home/me/simpapp/tuxconfig"
#               TUXDIR="/usr/tuxedo"
```

```
landingbj          LMID=simple

#Example:
#beatux            LMID=simple

*GROUPS
GROUP1             LMID=simple     GRPNO=1  OPENINFO=NONE

*SERVERS
DEFAULT:
                   CLOPT="-A"

simpserv           SRVGRP=GROUP1  SRVID=1

*SERVICES
TOUPPER
```

接下来，我们以此为引子，详细介绍 UBB 的各个配置部分及其相关参数。

6.2.1　*RESOURCES 段的配置

（1）IPCKEY 123456

IPCKEY 是 IPC 资源的标识符，它的取值为 32768～262143 之间的整数。

单机环境（SHM）下，IPCKEY 值标识公告板的入口地址，多机环境下（MP）它标识着 DBBL 进程的消息队列名，同一台主机的任何两个应用系统，IPCKEY 值不能相同。

（2）MASTER simple

MASTER 指定了作为主节点的逻辑主机标识。另外，还经常写为 MASTER SITE1，SITE2，这里 SITE1 为主节点，SITE2 为备份节点。

（3）MODEL SHM

单机模式（SHM）还是多机模式（MP）。单机配置指当前配置中只能有一台物理主机，即 MACHINES 段中只能有一个主机定义；MP 指当前配置中有两台以上的物理主机，即 MACHINES 段中包含两个以上主机。

MP 模式下会把 MASTER 节点的配置复制到非 MASTER 节点主机上，并实行统一管理。

（4）UID、GID、PERM

这 3 个参数控制着 Tuxedo 应用程序对系统 IPC 资源的存取权限。

UID 和 GID 是可以对该 Tuxedo 应用系统进行管理的用户 ID 和组 ID。如果不指定，默认为执行 tmloadcf 的用户 ID 和组 ID。

PERM 定义了 Tuxedo 应用程序对系统 IPC 资源的访问权限，默认值为 0666，即任何用户都具有对 Tuxedo 为该应用创建的 IPC 资源进行读写的权限。

（5）MAXACCESSERS 50

定义了公告板的最大容量，即同时可以容纳的客户机和服务器的数量，取值范围为 0～32768。默认值为 50。

（6）MAXSEREVERS 50

指定公告板可以容纳的最大服务器数量，取值在 0～8192 之间。

（7）MAXSERVICES 100

指定公告板可以容纳的最大服务数量，取值在 0～32768 之间。

（8）MAXGROUPS 100

指定公告板可以配置的组的数量，取值在 100～32768 之间。

（9）MAXGTT 100

定义最大并发全局事务数，取值在 0～32768 之间。

（10）MAXCONV 64

定义了最大并发会话数，取值在 1～32768 之间。

（11）LDBAL Y

指定是否启用负载均衡算法。

（12）MAXNETGROUPS

指定公告板中可以配置的网络组的数量。

（13）SYSTEM_ACCESS FASTPATH

可取值为 FASTPATH 或 PROTECTED，默认值为 FASTPATH。指示连接到服务器进程中的 Tuxedo 的内部库将以什么样的方式访问 Tuxedo 的内部表。

（14）SCANUNIT 10

设置系统健康检查的时间单位。

（15）SANITYSCAN 12

设置 BBL 对系统做健康检查的时间间隔，即每隔 SCANUNIT * SANITYSCAN 秒做一次健康检查。

（16）BBLQUERY 30

用于设置 DBBL 定期检查 BBL 状态的时间间隔。

（17）BLOCKTIME 6

设置客户端发出请求后，在得到响应之前，可以等待的最长时间为 BLOCKTIME * SCANUNIT，默认值为 6，即 60 秒。

6.2.2 *MACHINES 段的配置

（1）landingbj　LMID=simple

定义了主机名和逻辑主机标识。逻辑主机名和主机名一一对应，此逻辑主机名会被*RESOURCES 段中的 MASTER 参数和*GROUPS 段中的 LMID 参数引用。

（2）APPDIR

定义了 Tuxedo 应用程序的存放路径。

（3）TUXCONFIG

定义了二进制形式的 Tuxedo 配置文件的存放路径。

（4）TUXDIR

定义了 Tuxedo 在当前主机上的安装路径。

（5）MAXWSCLIENTS

指示最多可以有多少个工作站客户端连接到当前主机，取值在 0～32768 之间。

（6）ULOGPFX

指定系统日志在磁盘上的存储位置和文件名。

6.2.3　*GROUPS 段的配置

（1）GROUP1 LIMID = simple

指定一个组名和对应的逻辑主机，表明当前组内所有服务器都将部署到这台主机上。

（2）GPRNO = 1

为当前组指定一个编号，任何两个组编号不能相同。

（3）TMSNAME

指定事务监控器（Transaction Manager）进程的名称。

（4）OPENINFO

该串提供了打开 RM 连接所必须的参数。

（5）CLOSEINFO

该串指定关闭 RM 时需要提供的参数。

6.2.4　*NETWORK 段的配置

该段定义了网络相关的配置。如果*RESOURCES 段中的 MODEL 参数定义为 MP，并且 OPTIONS 参数中包含 LAN，则必须使用*NETWORK 段来对每一台主机的 tlisten 和 BRIDGE 进程监听的端口进行配置。

6.2.5　*SERVERS 段的配置

以示例 6-1 中的 simpserv 参数为例。

（1）SRVGRP = GROUP1

指定服务进程所属的组，这些组必须是在*GROUPS 段中定义过的。

（2）SRVID = 1

指定服务进程的编号，同一个组的两个进程，编号不能相同。

（3）CLOPT = “–A”

为服务进程指定命令行参数：“––”前的部分由服务器进程 main()函数捕捉处理，后面的部分由 tpsvrinit()捕捉处理。

“––”前面可配置的选项如下。

–A：初始化并公告服务进程中的所有服务。

–s：指定一个或多个服务名。

–e：指定一个文件名，服务进程输出到 stderr 的信息将被重定向到这个文件中。

–o：指定一个文件名，服务进程中输出到 stdout 的信息将被重定向到这个文件中。

–p：指示 Tuxedo 如何依据 SERVER 负载情况启动新进程和终止进程。

–r：该选项用于指示服务进程，把服务的执行情况记录下来，写到 stderr 中，可以使用 txrpt 命令（比如：txrpt -n landingbj -d06/19 -s00:03 -e14:28 <stderr）来分析这些执行记录。

–t：用于指定 WSL，GWTDMAIN 和其他服务进程与早期版本的兼容性。

（4）MIN MAX

启动服务器的最小和最大进程数。

（5）RQADDR =“Q1”

指定当前进程的请求队列名。若不指定队列名，Tuxedo 会按照组编号和进程编号的组合为它指定队列名。

（6）CONV = Y

Y 表示会话服务，N 表示不是会话服务。

（7）REPLYQ = Y

是否为当前进程创建单独的响应队列，建议取值 Y，此响应队列主要存放此服务作为客户端调用其他服务、其他服务的响应数据。

6.2.6 *SERVICES 段的配置

（1）LOAD = 50

定义当前服务的负载因子，LOAD 越大，说明服务的负载越大，处理时间越长。

（2）PRIO = 50

指定当前服务的优先级因子，PRIO 越大，说明当前服务的优先级越高，在服务器请求队列中优先出队的几率就越高。

（3）ROUTING = ACCOUNT_ID（参看示例 6-2）

指定服务使用的 DDR（数据依赖路由），这个规则必须是*ROUTING 段中定义的。

（4）SVCTIMEOUT = 60

以秒为单位，指定服务调用的超时时间，不指定表示永远不会超时。

（5）AUTOTRAN = Y

表示如果当前服务被调用时不在全局事务内，则自动开始一个全局事务。

6.2.7 *ROUTING 段的配置

在不同的组中配置相同的服务器，这样就可以为这些服务器提供的服务配置数据依赖路由，Tuxedo 收到调用请求后，就会根据请求缓冲区中某个字段的值，来决定把请求转发给哪个组中的服务器处理。

示例 6-2：

```
*ROUTING
ACCOUNT ID FIELD = ACCOUNT ID  BUFTYPE = "FML"
RANGES ="1000-4999:BANKB1,50000-7999:BANKB2,8000-109999:BANKB3"
```

6.3　域配置文件 DMCONFIG 及其内容

6.3.1　域（Domain）简介

随着企业信息化水平的不断提高，各类新兴业务的不断涌现，一个企业内部会有很多计算机应用系统，如 ERP、CRM、CALL CENTER、电子商务系统、大机遗留系统等，同时不同的企业之间要实现电子商务，他们之间的系统也要能够互相调用。在一个复杂的企业应用环境下，不同厂商的产品，不同应用系统之间要能够有效地互连，实现互操作，为企业构造一个紧密协作、集中管理的，同时又是一个高可靠、高可用、易于扩展的企业应用环境。

Tuxedo 是一个高度开放的系统，能够很容易地与别的应用系统实现互操作。为了有效实现与其他系统的互连，Tuxedo 提出了 Domain（域）的概念，将由很多台 （几百甚至上千）服务器共同组成的应用系统按功能或结构划分为不同的域，每个域独立地完成域内的操作，域间操作由域网关完成，从而提高每个域和整个系统的运行效率。

Tuxedo 的域特性把客户机/服务器模型扩展到多个独力自治的应用系统。一个域既可以是一组 Tuxedo 的应用程序，也可以是若干相关的应用服务和配置环境的组合，域同时也可能是一组运行在另一个非 Tuxedo 环境中的应用程序。

Tuxedo 和其他中间件的互操作也是利用域的概念来实现的。不同的 Tuxedo 应用域中的服务程序可以互相访问对方的服务，并且当一个交易同时执行多个应用域中的服务（即对于分布式事务处理）时，能够确保交易的完整性。同时，Tuxedo 系统可以指定哪些服务是可供外部应用域访问的，并可为这些服务设置访问控制表等安全认证手段，提高整个系统的安全性。

Tuxedo 对域的划分和管理类似于网络应用中划分子网的概念——将复杂的网络应用按功能或地域等因素划分为不同的子网，子网间以路由器连接，不同的网络协议通过网关透明地转换，从而提高网络效率并加强整个网络的可管理性。这一应用模式已被广泛证明是处理大范围、复杂应用的成功经验和事实上的标准，同时也是 Tuxedo 在多年大规模企业用户的实际应用中总结和开发的行之有效的中间件功能。而其他绝大多数中间件厂商尚无力涉足域的划分与管理，从而很难为超大规模的应用提供强有力的支持。

6.3.2　Tuxedo 域划分原则

Tuxedo 应用域是一个自治的、可独立配置管理运行的 Tuxedo 应用系统。根据应用的规模和企业组织的策略，将一个大的应用系统划分为几个独立的应用系统，不仅能够满足

组织管理的需要，而且能够独立调整监控，使系统性能达到最优。

影响应用域划分的因素有：应用规模；企业组织策略；应用安全范围；共享资源和服务的用户组。

Tuxedo/Domain 为各应用域之间提供了互操作的基础框架。应用域机制也为应用系统提供了与其他 OLTP 应用系统的互操作手段（通过 OSI-TP 标准协议）。

6.3.3 域（Domain）的功能

可扩展性和模块化指把一个系统划分为多个 Domain，可以很容易实现系统的扩展，而且每个 Domain 是互相独立的，对单个 Domain 的修改不会影响其他的 Domain。还可以根据系统的需要，加入新的 Domain。只要通过一些简单的配置，就可以实现 Domain 之间的互相访问。

位置透明就是 CLIENT 端不用知道它调用的一个 SERVICE 位于哪里，它可能在 LOCAL DOMAIN 上，或 REMOTE Domain 上，也可以位于 CICS 等其他系统上，或来自 WebLogic 上的一个 EJB。同一个 SERVICE，在 Domain 内外可以有不同的名字，管理员可以定义它们之间的对应关系。

支持跨越 Domain 的全局事务指 Tuxedo 支持一个全局事务跨越多个 Domain。

安全性指对 LOCAL Domain 中要被 REMOTE Domain 访问的 SERVICE 可以设置访问控制表，Domain 之间的数据传送支持加密和压缩，Domain 之间的连接支持认证。

6.3.4 Tuxedo Domain 的配置

Tuxedo Domain 的配置文件通常称为 DMCONFIG，是文本文件，在应用系统启动前要用 dmloadcf 把它编译成二进制文件 BDMCONFIG。

DMCONFIG 包括以下几个段，见表 6-3。

表 6-3

配置参数	含义和用法
DM_RESOURCES	指定总体域配置版本信息。本节中的唯一参数是 Version=string，其中 string 是一个字段，用户可以输入当前 DMCONFIG 文件的版本号，这个字段不由软件检查
DM_LOCAL_DOMAINS	定义该 LOCAL DOMAIN 的 DOMAIN ID（每个 DOMAIN 都有一个 DOMAIN ID），并对该 DOMAIN 所包含的所有 DOMAIN GATEWAY GROUP 进行定义 一个 Tuxedo 应用系统（DOMAIN）可以对应多个 DOMAIN GATEWAY GROUP，不同的 DOMAIN GATEWAY GROUP 可对应不同的 DOMAIN GATEWAY 类型
DM_REMOTE_DOMAINS	定义可以与 DOMAIN 中的 CLIENT、SERVER 进行互操作的 REMOTE DOMAIN，可以有多个 REMOTE DOMAIN
DM_LOCAL_SERVICES	定义 LOCAL DOMAIN 中可以被其他 REMOTE DOMAIN 访问的 SERVICE

续表

配置参数	含义和用法
DM_REMOTE_SERVICES	在这里配置 LOCAL DOMAIN 中可以调用的 REMOTE DOMAIN 中的 SERVICE
DM_ROUTING	在 DOMAIN 之间可以实现数据依赖路由（DDR），通过 DDR 把一个本地的请求路由到 REMOTE DOMAIN 上去 DOMAIN 之间的 DDR 规则在 DM_ROUTING 中设置，DOMAIN GATEWAY 通过在这里指定的规则把请求发到相应的 REMOTE DOMAIN 中去
DM_ACCESS_CONTROL	为 LOCAL SERVICE 设置访问控制表，定义哪些 REMOTE DOMAIN 可以访问这些 LOCAL SERVICE

如果要配置 Domain 还需要对 UBB 中的配置进行修改，即在 UBBCONFIG 中要对 DMADM、GWADM、GWTDOMAIN 3 个 SERVER 进行配置。

6.4　日志文件 ULOG

User Log（ULOG） 是由 Tuxedo 系统在应用处于运行时产生的一个日志文件，主要记录 Tuxedo 系统运行时产生的一些有用的信息。应用客户端和服务端把日志记录到这个文件中，每天产生一个新的文件，每个机器都有不同的日志记录。在集群中日志可以被共享给其他的远端系统使用。ULOG 使定位问题的工作变得简单，直观。用户可以用 ULOG 去识别引起系统或者是应用运行失败的原因。通常在日志中较早产生的日志提供的信息对用户的帮助远大于较晚产生的信息。

ULOG Message 示例如下。

示例 6-3：

```
151550.landingbj!BBL.28041.1.0: LIBTUX_CAT:262: std main starting
151550.landingbj!BBL.28041.1.0: LIBTUX CAT:358: reached UNIX limit on ...
151550.landingbj!BBL.28041.1.0: LIBTUX CAT:248: fatal: system init...
151550.landingbj!BBL.28040.1.0: CMDTUX CAT:825: Process BBL at SITE1 ...
151550.landingbj!BBL.28040.1.0: WARNING: No BBL available on site SITE1.
```

ULOG 日志提供了比较完整的信息，从管理员的角度来说这就是相当于为解决问题指明了方向。

第 3 篇

实　施　篇

第 7 章　Tuxedo 应用的部署模式

Tuxedo 使用 Domain 来组织应用程序。通常情况下，一个应用程序由一个 Domain 组成，这种组织称为“单域”模式（Single-Domain Model）。在某些情况下，一个复杂的应用程序可能由多个 Domain 组成，这种组织模式称为“多域”模式（Multi-Domain Model）。

在“单域”模式下，一个应用程序既可以部署在一台物理主机上，又可以部署在多台使用高速局域网连接在一起的主机上。如果应用程序只部署在一台物理主机上，这种模式称为“单机”模式（Single Host Model，SHM）；如果应用部署在多台主机上，这种部署模式称为“多机”模式（Multi-Processor，MP）。

在“多域”模式下，一个应用程序由若干个逻辑上相对独立的 Domain 组成。这些 Domain 通过网关进程连接在一起，其中一个 Domain 既可以把自己实现的服务发布给其他 Domain，也可以从其他 Domain 中导入服务。多域中的每个域既可以使用单机部署，也可以使用多机部署模式。

单机模式、多机模式和多域模式是 Tuxedo 应用程序的 3 种组织模式。

7.1　单机 SHM 模式

单机模式是使用最多，也是最简单的一种组织模式。在这种模式下，所有业务处理进程，Tuxedo 系统进程和管理进程都部署在同一台物理主机上，这台主机既担负着域的管理任务，又担负着业务的处理任务。单机模式的网络拓扑结构比较简单，UBB 配置文件也很简单，如下所示。

示例 7-1：

```
*RESOURCES
IPCKEY          96338
DOMAINID        simpapp
MASTER          SITE1
MAXACCESSERS    200
MAXSERVERS      120
MAXSERVICES     350
MODEL           SHM
LDBAL           N
*MACHINES
DEFAULT:
        TUXDIR="/tuxedo/tuxedo10.0" #相关目录需要更改为用户自己的
        APPDIR="/tuxedo/service app/bin"
        TUXCONFIG="/tuxedo/info/tuxconfig"
```

```
        ULOGPFX="/tuxedo/log"
        MAXWSCLIENTS=100
landingbj  LMID=SITE1
#landingbj 是主机名，UNIX 可通过 uname -n 获得，LIMD 表示主机逻辑 ID
*GROUPS
GROUP1  LMID=SITE1 GRPNO=1  OPENINFO=NONE
GROUP2  LMID= SITE1  GRPNO=2 OPENINFO=NONE
*SERVERS
DEFAULT:RESTART=Y MAXGEN=10 GRACE=3600
simpserv   SRVID=1 SRVGRP=GROUP1 MIN=5 MAX=10
WSL        SRVID=10  SRVGRP=GROUP2
         CLOPT="-A -t -- -n //192.168.21.128:7110
-m 10 -M 20 -x 10 -c 1024"
SERVICE
TOUPPER   PRIO=50 LOAD=50
```

UBBCONFIG 文件具有如下关键字段需要注意（详细内容请参考第 6 章）。

```
*RESOURCE
```

定义了 Tuxedo 应用程序中使用到的域级参数，核心参数如下。

IPCKEY：Tuxedo 公告板（BB，Bulletin Board）的唯一 IPC 标识符。

MASTER：管理服务器，也称为主节点（Master）的逻辑名字。

MAXACCESSERS：最大的域访问量。

MAXSERVERS：一个域中可以配置的最大服务进程数。

MAXSERVICES：一个域中可以发布的最大服务数。

MODEL：指定单机域还是多机域。SHM=单机，MP=多机。

LDBAL：是否启用负载均衡。Y=启用负载均衡。

```
*MACHINES
```

指定域中包含的计算机。在 SHM 模式中，该节点只能包含一台计算机。在该节点中，必须设置 TUXCONFIG，并且它的值必须与 TUXCONFIG 环境变量中的值相同。

```
*GROUPS
```

Oracle Tuxedo 允许根据业务需要把后台服务分成组，例如，账务服务可以定义在“FINAN_GRP”组中，人力资源服务可以定义在“HR_GRP”中。一个组中的所有服务必须部署在*MACHINE 段中定义的一台计算机上。组同样也是消息路由以及服务迁移的单位。在 XA 事务环境中，组还与资源管理器相关联。

```
*SERVERS
```

定义域中部署的服务进程。它必须指定一个服务组。在上面的例子中配置了一个 simpserv 服务进程，属于 GROUP1 组。其中 DEFAULT 关键字用于设置一些通用的参数，这些参数对 DEFAULT 以下所有服务进程都有效。本例中设置了 RESTART、MAXGEN 和

GRACE 3 个参数，表示 simpserv 和 WSL 两个进程都是可重启的，即当 Tuxedo 系统发现它们处于 DEAD 状态时，就会尝试着重新启动它们。

SHM 应用的特点是单机模式支持进程级的伸缩性和容错。

按照上面的这个配置，Tuxedo 在启动 SIMPAPP 时，会启动 5 个 simpserv 实例。当压力增大时，用户可以启动更多的 simpserv 实例，但最多不超过 10 个；当压力减小时，用户可以终止 simpserv 进程实例，但最少要保留 5 个。在系统运行过程中，如果某些 simpserv 实例异常终止了，Tuxedo 系统会把它们隔离起来，然后把客户请求派发给处于活动状态的实例去处理。

如果一个进程只启动了一个实例，那么它就不具备进程级容错功能。比如，本例中的 WSL 进程，它只启动了一个实例，监听着 landingbj 主机的 7110 端口，如果这个进程死了，在它被 Tuxedo 系统重新启动之前，所有远程客户端将不能再连接到 Tuxedo 服务器上来做交易。

单机模式不支持主机级容错。如果 landingbj 主机因硬件错误不能再运行，那么整个 SIMPAPP 将不能再对外提供服务。因此，大多数单机应用通常借助硬件 HA 方式来提供主机级容错功能。

7.2 多机 MP 模式

在多机模式下，一个 Tuxedo 应用程序需要部署在多台物理主机上，这些主机通过高速局域网连接在一起，并在 Tuxedo 系统的协调下，共同完成特定的任务。

MP 应用的构成：如图 7-1 所示，MP 的 Master 节点上面要运行 DBBL、BBL、Bridge、tlisten（以备 Master 迁移时使用）4 个系统进程。MP 的其他节点上面要运行 tlisten、Bridge、BBL3 个系统进程。此外，每个节点上都运行着若干个本地客户进程（Clients）和应用服务进程（Servers）。每个节点上的 BBL 都维护着一个本地公告板（Bulletin Board，BB），Master 节点上的 DBBL 负责协调使所有公告板的数据保持一致。

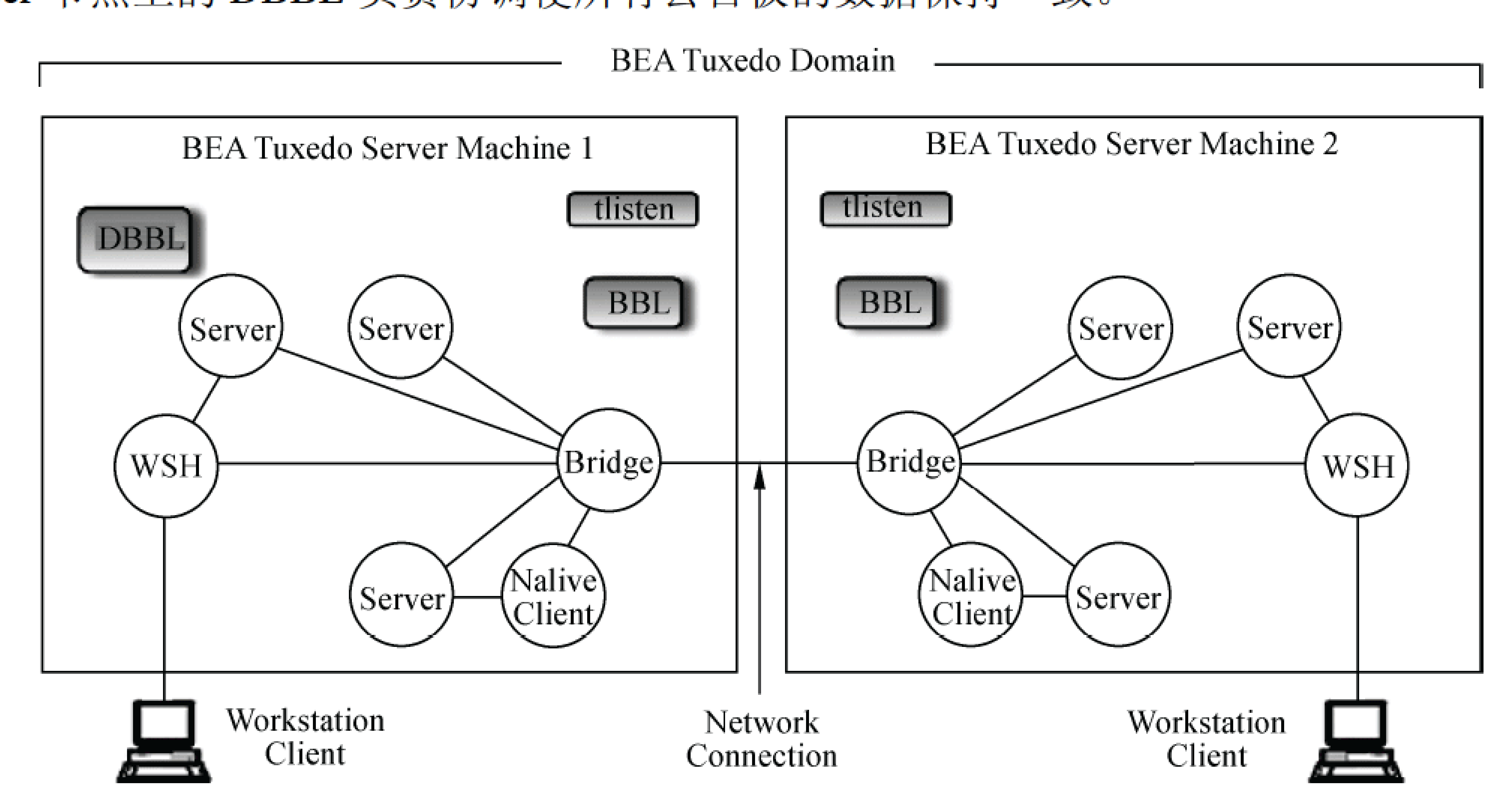


图 7-1

各个节点之间通过桥进程（Bridge）连接在一起，所有节点之间的消息交换都通过 Bredge 进程来完成。在每一个节点的公告板中既可以找到本地服务的调用入口，也可以找到远程服务的调用入口。当客户机要调用一个本地服务时，它可以直接从公告板中找到提供这个服务的服务器的 IPC 请求队列，向这个队列发送请求；当客户机要调用一个远程服务时，它从公告板中找到本地 Bridge 进程的消息队列，然后把请求消息放到里面，Bridge 会把请求消息路由到适当的远程服务器的请求队列，远程服务器处理完请求后，再将响应结果通过 Bridge 传回给发出请求的客户机。这一切对客户机是透明的，也就是说客户机无需知道哪一个服务是本地的，哪一个服务是远程的。

Master 节点维护着整个应用程序配置文件的主拷贝（TUXCONFIG-original），在执行 tmboot 启动应用程序时，DBBL 会与成员节点上的 BBL 进行通信，并把配置文件复制到各个成员节点上，BBL 再根据配置文件（TUXCONFIG-copy）更新它们维护着的公告板。

从管理控制台（Administrative Console）上发出的对成员节点的管理指令，也是通过 BRIDGE 进程转发到远程的，远程节点的 BBL 进程收到管理指令后，修改公告板，完成管理任务。

tllisten 是一个运行在 Non-Master 节点上的后台网络监听进程，它接收来自 Master 节点的管理指令，然后启动本地的 BSBRIDGE 进程。因此在执行 tmboot 启动应用程序之前，所有节点上必须先启动 tlisten 进程。事实上，在启动应用程序的过程中，Master 节点上的 tlisten 并没有发挥任何作用，因此没有必要启动它。尽管如此，还是推荐把 Master 节点上的 tlisten 也启动起来，以防将来因为系统故障而把 Master 节点降级为 Non-Master，把原来的 Non-Master 节点提升为 Master。

表 7-1 对 DBBL、BBL、Bridge、tlisten 4 个进程的用途做一个总结。

表 7-1

进程	说明
DBBL	Distinguished Bulletin Board Liaison（DBBL），作用是记录所有 BBL 的状态，并负责与它们联络，保持各个主机上的公告板的数据同步
BBL	Bulletin Board Liaison（BBL），作用是维护本地公告板，记录所有服务器和服务的状态，定期对 Tuxedo 系统作健康检查，与 DBBL 通信保持公告板上的数据的同步
Bridge	Bridge 进程被称为桥进程，它的作用是负责各个节点间的数据通信。这个进程在 tmboot 时会自动启动，不需要在 UBBCONFIG 文件中对它进行配置
tlisten	tlisten 是一个在后台独立运行的网络监听进程，它的作用是从管理控制台、公告板或命令行接收命令，然后启动 BBL、Bridge 等管理进程

MP 应用的启动流程相对比较复杂，图 7-2 展示了一个两节点 MP 应用程序的启动过程。具体启动步骤如下。

（1）在 Master 机器上执行 tmboot 启动服务（在非 Master 机器上已启动 tlisten 进程）。

（2）在 Master 机器上首先启动 DBBL 进程。

（3）启动 BBL 管理进程。

（4）启动 Bridge 进程，Bridge 进程与非 Master 机器建立连接，此时非 Master 机器上所有服务都还没有启动（除了 tlisten 进程外）。

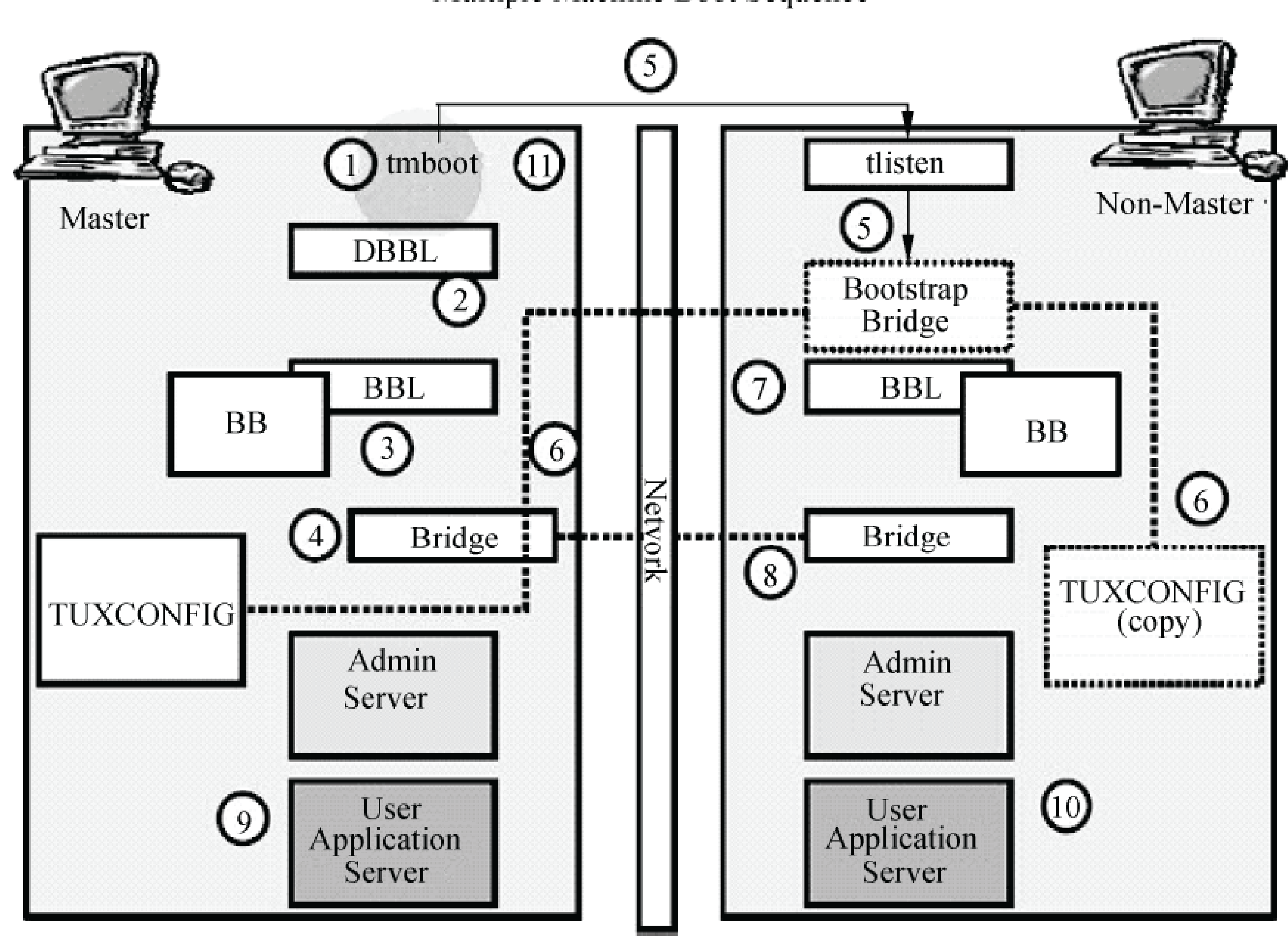


图 7-2

（5）tmboot 连接到非 Master 机器上的 tlisten 进程，tlisten 进程启动 BSBRIDGE（bootstrap Bridge）进程。

（6）BSBRIDGE 进程连接到 Master 机器上的 Bridge 进程，并向 Master 索求 TUXCONFIG 配置文件，TUXCONFIG 配置文件传递到非 Master 机器上。

（7）启动远端机器的 BBL 创建 BB。

（8）启动远端机器的 Bridge 进程，同时 BSBRIDGE 进程终止，此时两台机器的 Bridge 进程建立了连接，此后两台机器之间的通信主要通过 Bridge 进程。

（9）启动 Master 机器上的其他进程。

（10）启动非 Master 机器上的其他进程。

MP 应用的配置也相对稍微复杂一些，如下所示为一个简单的 MP 配置。

示例 7-2：

```
*RESOURCES
IPCKEY                  309992
#SECURITY               APP PW
MASTER                  SITE1 ,SITE2
MAXACCESSERS            450
MAXSERVERS              350
MAXSERVICES             350
MODEL                   MP
OPTIONS                 LAN
ENCRYPTION REQUIRED     N
LDBAL                   Y
BLOCKTIME               300
```

```
*MACHINES
vmLinux         LMID=SITE1
                APPDIR="/home/landingbj/bin"
                ULOGPFX="/home/landingbj/bin/log/ULOG"
                TUXCONFIG="/home/landingbj/bin/tuxconfig"
                TUXDIR="/home/tuxedo/bea/tuxedo8.1"
                MAXWSCLIENTS=200
"LANDINGBJ"     LMID=SITE2
                APPDIR="C:\landingbj\bin"
                TUXCONFIG="C:\landingbj\bin\tuxconfig"
                TUXDIR="C:\bea\tuxedo8.1"
                UID=0
                GID=0
                MAXWSCLIENTS=200
*GROUPS
GRPM            LMID=SITE 1     GRPNO=60
GRPGZQH         LMID=SITE 2     GRPNO=100
*NETWORK
SITE1           NADDR="//192.168.250.101:6001"
                NLSADDR="//192.168.250.101:6002"
SITE2           NADDR="//192.168.250.103:6001"
                NLSADDR="//192.168.250.103:6002"
*SERVERS
DEFAULT:
                CLOPT="-A" SYSTEM ACCESS=FASTPATH
WSL              SRVGRP=GRPM      SRVID=100 RESTART=Y MAXGEN=5  CLOPT="-A -- -n
//192.168.250.101:12345 -m 3 "
simpserv         SRVGRP=GRPM      SRVID=1
procclient       SRVGRP=GRPGZQH   SRVID=10
```

MP 配置文件并不复杂，在编写的时候要注意以下几点。

（1）在*RESOURCES 段，MODEL 必须为 MP，表示当前是一个 MP 应用。

（2）OPTIONS 必须配置为 LAN，表示当前应用跨越网络，部署在多台服务器上；同时也可以加上 MIGRATE 选项，表示可迁移。

（3）MASTER 参数中可以加入一个备用 Master 节点。

（4）把 LDBAL 参数设置为 Y，表示激活负载均衡算法。

（5）在*MACHINES 段中，必须加入参与计算的每一个节点的配置信息。

（6）同时可以使用 NETLOAD 为每个节点指定网络负载因子，这个参数表示把一个请求从本地转发到远程去处理要付出的代价，取值越大表示有更多的请求会在本地处理。

此外还有两个与 MP 模式相关的重要参数，它们是 BBLQUERY 和 DBBLWAIT。

（7）BLQUERY 用于设置 DBBL 定期检查 BBL 状态的时间间隔，默认值为每隔 300 秒检查一次。如果 BBL 没有回应，则 DBBL 就会认为它所管理的 Tuxedo 系统不再是 MP 的一部分，并且将它隔离出去。

（8）DBBLWAIT 是 DBBL 在给某个 BBL 发送消息后，等待它作出回应的最长时间，默认为最多等待 20 秒，如果 BBL 没有回应，DBBL 就认为它们已经死了。

MP 应用的特点是容错能力强与负载均衡。

MP 应用程序具有容错能力强、可靠性高的特点。首先，Tuxedo 运行管理机制提供了对软件错误的自动监控和修复功能。每一个管理进程（如 DBBL、BBL、Bridge 等），以及处理业务逻辑的服务进程都被 Tuxedo 系统自动监控起来，当某个进程出现错误时，Tuxedo 都会自动尝试着去修复它，而不需要人工干预。

在 MP 应用程序中，DBBL 会定期向各个节点的 BBL 进程发送询问消息，BBL 收到询问后，会向 DBBL 发送类似于“我还活着”的回应消息，这种机制被称为“心跳监测”。如果 DBBL 在特定的时间内得不到 BBL 的回应，就会认为 BBL 已经死了，并试着去重新启动它。如果该节点上的 BBL 不能被启动，DBBL 就会把该节点标记为“Partitioned”的状态，然后自动隔离起来。

Master 节点上的 BBL 反过来也会监控 DBBL 的状态，必要时也会尝试着去启动它。如果 DBBL 不能被启动，就需要管理员手工地将 DBBL 迁移到备用节点上。每个节点上的业务进程都可以配置可重启属性。在例行健康检查时，当 BBL 检查到某个进程死了，也会试着去重新启动它。

其次，MP 应用程序支持主机级容错。即便是构成 MP 的某几个节点死了，应用程序仍能够不间断地对外提供服务。MP 模式的主机级容错特性有别于硬件 HA 容错。在硬件 HA 模式下，只有当主节点发生故障时，备用节点才会接替它的工作，否则备用节点始终处于空闲状态。在 MP 模式下，所有的节点都参与计算，某些节点发生故障时，会被自动隔离出去。

再次，MP 应用程序的远程客户机访问可以通过设置 WSNADDR 环境变量来提高容错能力。假设 1 个 MP 应用由 3 个节点组成，它们的地址分别为 IP1、IP2 和 IP3，每个节点上都运行着 WSL 进程，端口分别为 PORT1、PORT2 和 PORT3，则客户端可以通过下面的方式来设置 WSNADDR，以提高容错能力：

```
SET WSNADDR=//IP1:PORT1,//IP2:PORT2,//IP3:PORT3
```

在这种情况下，客户机会依次尝试着使用这 3 个地址来连接服务器，如果第一个连不通就会尝试第二个，第二个连不通就会尝试第三个，如果都连不通，tpinit()才会报错。

另外在 MP 模式下，负载均衡功能默认是打开的，即 LDBAL 取值为 Y。这样，所有节点都会均匀分摊客户机请求造成的负载。然而，节点之间并不会相互通知彼此的负载信息，即每个节点并不知道其他节点负载的准确值，只能作一个大致的估计。因此，不同的远程节点对同一个节点的负载估计完全可能是不一样的。

对于同一个服务请求，假设在本地处理的负载为 LOAD，转发到远程处理要付出的额外代价为 NETLOAD，则每当本地处理了（1+NETLOAD/LOAD）个请求后，就会把一个请求转给远程处理。假设 LOAD=50，NETLOAD=100，则每当本地处理 3 个请求之后，就会把 1 个请求转给远程处理。

可以在*MACHINES 段中为每个节点指定 NETLOAD，也可以通过 TMNETLOAD 环境变量来指定，取值越大，就会有更多的请求在本地处理。

在远程客户端，按照下面的方式设置 WSNADDR 环境变量也可以达到 3 台服务器之间负载均衡的目的：

```
SET WSNADDR=(//IP1:PORT1^|//IP2:PORT2^|//IP3:PORT3)       #Windows
```

```
WSNADDR=(//IP1:PORT1|//IP2:PORT2|//IP3:PORT3)              #UNIX
```

每一次客户机执行到 tpinit()调用时，都会从 3 个服务器地址中任选一个来建立连接。

客户机可以和任何节点建立连接，如果该节点没有提供客户机请求的服务，Tuxedo 就会把请求转发到其他提供了该服务的节点上去处理，如果多个节点都提供了同一个服务，那么 Tuxedo 就会把请求送到负载最小的那个节点去处理。

7.3　多域模式

企业的 IT 部门通常会按功能、地理位置和机密等级来把企业的 IT 系统规划成若干个子系统，这些子系统在管理上是完全独立的，它们既可以独立工作，又可以通过网络连接在一起，实现协同工作和应用集成，形成一个统一的信息平台，为不断变化和增长的企业业务提供服务。随着时间的推移，各类子系统会越来越多，如何把这些子系统很好地整合起来，是企业 IT 部门必须应对的挑战。

Tuxedo 引入了域的概念。域是 Tuxedo 分布式应用程序的一种组织方式和管理单元，一个域就是一个独立的应用系统，它既可以由单个节点构成（SHM 模式），又可以由多个节点构成（MP 模式），它的配置通过一个 TUXCONFIG 文件来描述。

在企业计算环境中，一个 Tuxedo 域可以使用特定网关去连接其他域，实现业务集成。一个 Tuxedo 域可以使用 TDOMAIN 网关去连接另一个 Tuxedo 域，也可以使用 TDOMAIN/WTC 去连接一个 WebLogic 域。如果企业计算环境中存在 TOPEND 域或主机应用，Tuxedo 域也可以使用 TEDG 或 TMA 网关去连接它们。

一个域可以把其他域的服务导入到本地，也可以把本地服务发布给其他域。对于客户端而言，本地服务和远程服务是完全透明的，即本地客户端可以像调用本地服务一样去调用一个远程服务，反之亦然。域和域之间支持安全上下文传递，可以为跨域的服务调用设置安全验证和访问控制。域和域之间还支持事务上下文的传递，多个跨域的操作可以纳入到同一个全局事务中来进行事务控制。在 Tuxedo 和 WebLogic 之间，还可以跨域实现单点登录（Single Sign On，SSO）。

在 MP 模式下，所有节点必须位于同一个局域网中，而在多域模式下，连接两个域之间的链路可以是不可靠的广域网，如图 7-3 所示。如果要跨越网关传递大量数据，可以在网关上配置压缩特性；如果要通过网关传递重要数据，可以在网关上配置加密特性。

Tuxedo 提供了 3 类域网关来连接不同的系统，它们是 TDOMAIN、TOPEND 和 TMA。根据使用协议和目的的不同，可以把 TMA 网关分为 TCP、SNA、OSI TP 3 类。

TDOMAIN 网关是基于 TCP/IP 设计的，它的进程名是 GWTDOMAIN，常用于连接其他 Tuxedo 域和 WebLogic 域。在连接 WebLogic 时，WebLogic 一端使用的是 WTC。TOPEND 网关也是基于 TCP/IP 网络设计，进程名是 GWTOPEND，它为 Tuxedo 系统连接 TOPEND 系统提供支持，从 9.0 开始，Tuxedo 将不再支持这个网关。TMA for Mainframe TCP 网关进程名是 GWIDOMAIN，它为 Tuxedo 系统连接 IBM OS/390 的 CICS 和 IMS 系统提供支持。TMA for Mainframe SNA 网关的进程名是 GWSNAX，它为 Tuxedo 系统连接 IBM OS/400、OS/390 CICS 和 IMS、VSE/CICS 等使用 SNA 网络的系统提供支持。TMA for Mainframe OSI TP 网关进程名是 GWOSITP，它为 Tuxedo 系统连接其他 OSI TP 系统（如

IBM CICS、Ecina、Tong/Easy 等）提供支持。

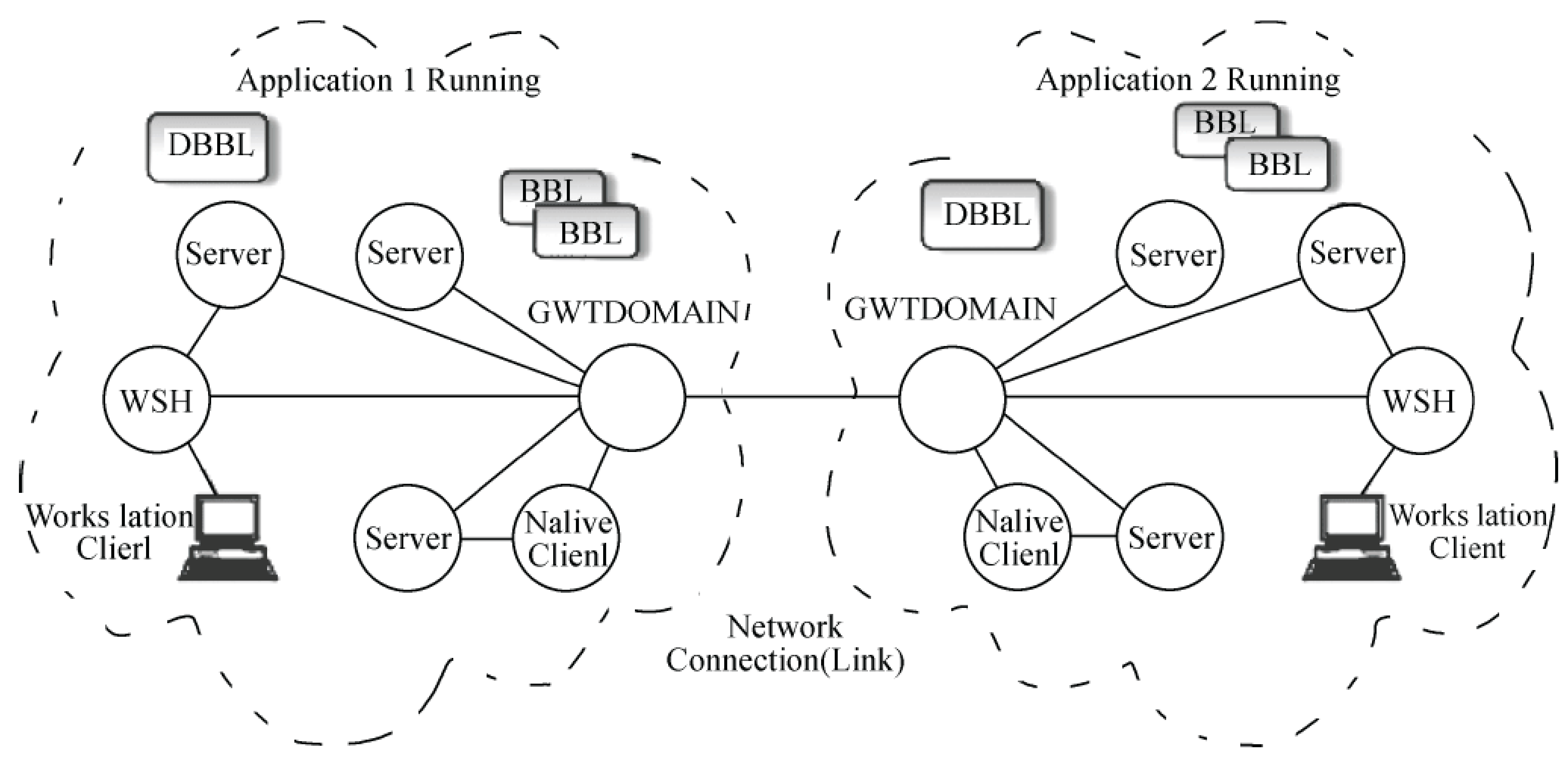


图 7-3

连接不同类型的远程域需要使用不同的网关（如连接 WebLogic 使用 GWTDOMAIN，连接 TOP END 域使用 GWTOPEND），但对于同一类型的多个远程域来说，既可以使用一个网关也可以使用多个网关。如果 1 个本地 Tuxedo 域要和 3 个远程 Tuxedo 域建立连接，那么有两种方案可以选。第一种方案是在本地域中使用 1 个网关去连接 3 个远程域，第二种方案是在本地域使用 3 个网关分别去连接 3 个远程域。第一种方案的优点是网关数量少，易于管理和监控，缺点是网关压力大，没有办法针对每一个远程域设置个性化的连接策略。第二种方案的优点是可以设置个性化连接策略，网关压力小，多网关之间可以设置 Failover，缺点是网关进程多，不易于管理。

每个域都有一个域管理进程 DMAMD，它管理着域的配置文件 BDMCONFIG 和网关组。每个组有一个网关管理进程 GWADM 和一个网关进程，GWADM 管理着网关。网关进程负责域之间的通信，它可以把远程域的服务导入到本地，并在 BB 中进行发布，使本地客户端可以调用它们，如图 7-4 所示。

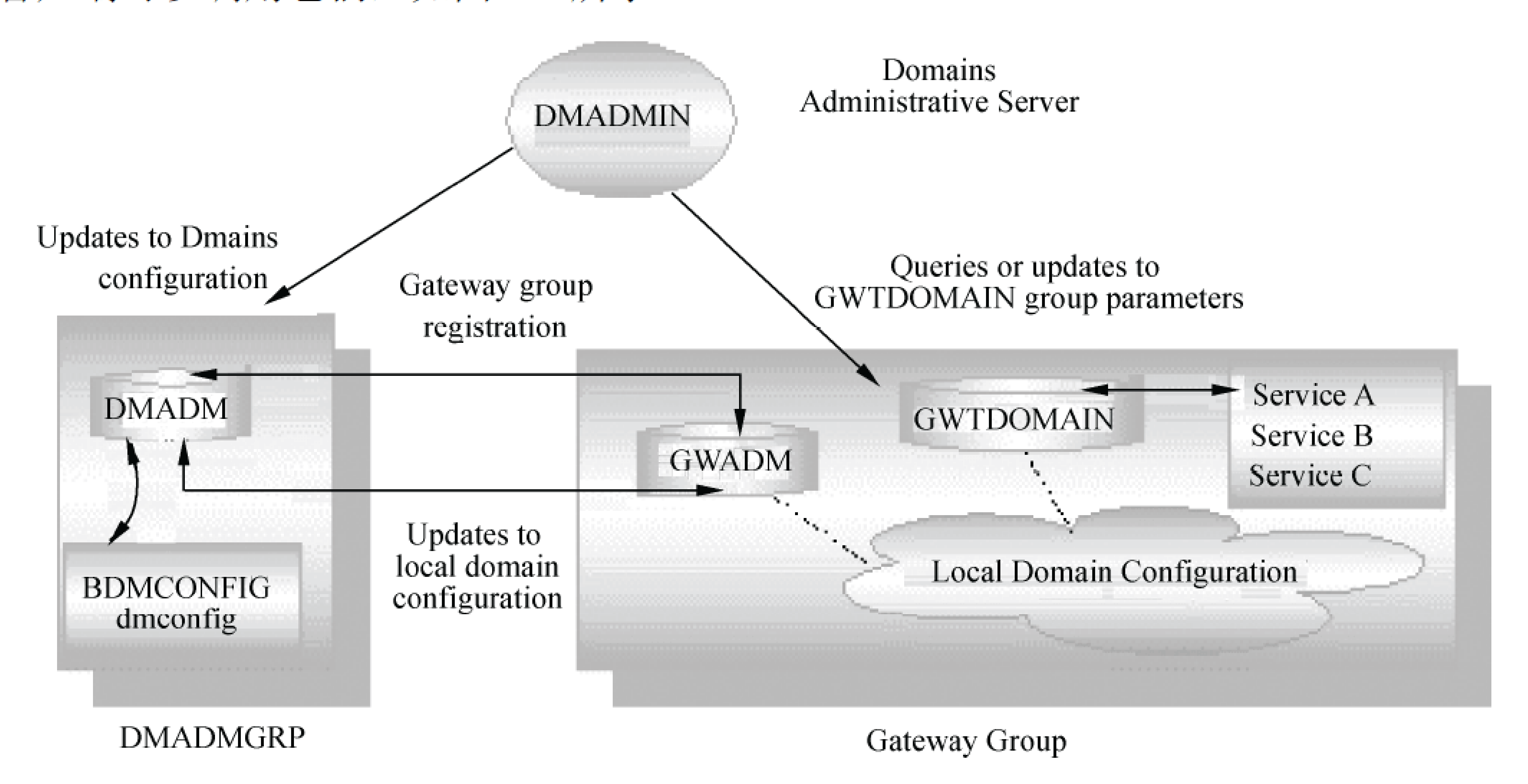


图 7-4

表 7-2 详细地描述了这些进程的作用。

表 7-2

进程	说明
DMADM	域管理进程，管理配置文件 BDMCONFIG 和网关组，它发布了一个同名的注册服务，每个网关组的管理进程 GWADM 在初始化时，都要调用这个服务来注册网关
GWADM	运行时域网关管理服务器进程，它从 DMADM 服务器上获取信息，并定期告诉它“我还活着！”
GWTDOMAIN	TDOMAIN 的网关进程，负责导入远程域的服务，并在本地域的 BB 中发布它们。它是本地域对远程域的访问点，负责域之间的可靠消息传递、数据加密、压缩和安全认证等

如果一个域有 *n* 个网关组，那么域管理进程、网关管理进程和网关进程的数量总和为 $2n+1$ 个。多域应用中的每个域可以独立启动，在什么时候和远程域建立连接，将由域网关上配置的连接策略决定。

关于多域应用基本配置，Tuxedo 使用一个单独的配置文件 DMCONFIG 来保存域的配置信息。DMCONFIG 和 UBBCONFIG 文件非常类似，也有 ASCII 和二进制两个版本。ASCII 版由管理员编写，在启动之前，必须使用 dmloadcf 把它转换成二进制的形式，并通过 BDMCONFIG 环境变量指向它。

DMCONFIG 文件由若干个段组成，下面按顺序对它们的关键点进行提示。

```
*DM_RESOURCES
```

这个段定义全局的域配置信息，目前只有一个 VERSION 参数。管理员可以用这个参数来标记域配置文件版本，取值是一个字符串，除此之外没有其他意义。这个段不是必须的，可以不定义。

```
VERSION=SIMPAPP V1
*DM_LOCAL
```

这个段的别名是*DM_LOCAL_DOMAINS，用于定义本地域网关访问点。每个域网关对应着一个访问点，如果本地域有多个网关，则必须在这里定义多个访问点。下面的例子定义了一个本地域网关访问点 DOM1。

```
DOM1   GWGRP=LGWGRP    TYPE=TDOMAIN    DOMAINID="DOM1"
```

在这个段中，GWGRP、TYPE、DOMAINID 参数是必须的，其他参数都是可选的。GWGRP 指定了网关进程所属的组名 LGWGRP，它必须在 UBB 文件中定义过。DOMAINID 的别名是 ACCESSPOINTID，它用于定义网关标识，在配置安全特性和 Failover 时，会用到这个标识。TYPE 用于定义网关的类型，可取值为 TDOMAIN、TOPEND、SNAX、OSITP 和 OSITPX。

```
*DM_REMOTE
```

这个段的别名是*DM_REMOTE_DOMAINS，用于定义远程域网关访问点。下面的例

子定义了一个远程域网关访问点 RMDOMAIN。

```
RMDOMAIN    TYPE=TDOMAIN    DOMAINID="RMDOMAIN"
```

这个段中，DOMAINID 和 TYPE 参数是必须的，其他参数是可选的。

```
*DM_TDOMAIN
```

这个段为本地网关和远程网关指定监听地址和端口。远程网关通过本地网关的监听地址来建立连接和发送请求，本地网关通过连接到远程网关的监听地址上去发送服务请求。下面的例子定义了两个网关访问点的监听地址。

```
DOM1    NWADDR="//192.168.21.128:2507"
RMDOMAIN    NWADDR="//192.168.21.127:3186"
```

在这个段中，只有 NWADDR 是必需的，其他的可选参数还有 NWDEVICE（指定使用的网络设备）、CMPLIMIT（指定压缩阀值）、TCPKEEPALIVE（指定是否保持 TCP 连接）。

```
*DM_ACCESS_CONTROL
```

这个段用于定义访问控制列表（ACL），以控制哪些远程域可以请求本地的服务。下面的例子定义了一个名为 MY_ACL 的访问控制列表。

```
MY_ACL      ACLIST=RMDOMAIN,RMDOMAIN1
*DM_EXPORT
```

这个段的别名是*DM_LOCAL_SERVICES，用于定义本地域导出的服务，这些服务可被远程域导入。在默认情况下，所有的本地域的服务都是可以导出的，这些服务继承了在 TUXCONFIG 中定义的属性，如 LOAD、PRIO、AUTOTRAN、ROUTING、BUFTYPE 和 TRANTIME。下面的例子是为本地域导出的服务 TOLOWER 定义了 ACL 和服务别名。

```
TOLOWER    RNAME=LOWERCASE  ACL=MY_ACL
```

这个例子说明，只有来自 RMDOMAIN 和 RMDOMAIN1 网关的请求才能调用 TOLOWER 服务，来自其他网关的请求会被拒绝。RNAME 为本地服务提供了重命名功能，当远程域导入 TOLOWER 时，服务名不再是 TOLOWER，而是 LOWERCASE。

这个段用于定义本地服务的参数有 RNAME、LDOM、ACL 和 CONV。RNAME 用于指定导出服务的别名。LDOM 的别名是 LACCESSPOINT，用于指定导出网关。ACL 用于指定一个*DM_ACCESS_CONTROL 中定义的 ACL 名，CONV 可取值 Y 和 N，用于指定导出的服务是否为会话类型。

```
*DM_IMPORT
```

这个段的别名是*DM_REMOTE_SERVICES，用于定义从远程域中导入的服务。下面的配置用于从远程域导入 TOUPPER 服务。LDOM 的别名是 LACCESSPOINT，它指定了本地网关访问点是 LAPP。RDOM 的别名是 RACCESSPOINT，它指定了 TOUPPER 的远程网关访问点是 UAPP。当本地调用 TOUPPER 时，服务请求将由 LAPP 传递给 UAPP。

```
TOUPPER   LDOM=LAPP    RDOM=UAPP  RNAME=UPPERCASE
```

这个段中用于定义远程服务的参数还有 CONV、LOAD、RNAME、ROUTING 等。与*DM_EXPORT 段的同名参数相同，RENAME 用于指定导入服务的别名，CONV 用于指定服务是否为会话类型，另外，LOAD 用于指定负载因子，ROUTING 用于指定路由规则。

```
*DM_ROUTING
```

这个段用于定义多个域之间的数据依赖路由规则，这些规则可以被* DM_REMOTE_SERVICES 段中的服务引用。下面的例子定义了一个路由规则 ON_ACCOUNT，这个规则被 ACCT_BAL 服务引用。

```
*DM_REMOTE_SERVICES
     ACCT BAL LDOM=JDG ROUTING=ON ACCOUNT
*DM_ROUTING
     ON_ACCOUNT BUFTYPE="FML32" FIELD=ACCOUNT_ID
RANGES="0-9999: MDG1, *: MDG2"
```

ACCT_BAL 是一个导入服务，MDG1 和 MDG2 两个域都提供了该服务，它在本地的网关访问点是 JDG，请求缓冲区类型是 FML32。路由规则 ON_ACCOUNT 的作用是，如果请求缓冲区中的 ACCOUNT_ID 字段取值在 0～9999 之间，则本地域将通过 JDG 把请求转发给 MDG1 域，否则将通过 JDG 转发给 MDG2 域。

*DM_ROUTING 与 UBBCONFIG 文件中的*ROUTING 的用法基本是一样的，差别仅在于 UBBCONFIG 中*ROUTING 的 RANGES 参数中，取值范围对应的是组名，而在 DMCONFIG 中*DM_ROUTING 中取值范围对应的是远程网关访问点。

7.4　各种模式的比较

单机模式是用得最多，也是最简单一种组织模式。在这种模式下，所有业务处理进程、Tuxedo 系统进程和管理进程都部署在同一台物理主机上，这台主机既担负着域的管理任务，又担负着业务的处理任务。

多机模式其实就是 Tuxedo 系统的集群模式。在这种模式下，主机通过高速局域网连接在一起，并在 Tuxedo 系统的协调下，共同完成特定的任务。主机之间支持数据依赖路由、负载均衡和容错功能。

多域模式是 Tuxedo 推出的一种更为灵活的分布式应用程序组织方式。多域模式可以把业务功能、地理位置和机密等级各不相同的应用程序很好地组织在一起，使它们可以相互调用对方服务，实现业务渗透。

Tuxedo 提供的域网关支持别名功能，可以使服务以别名的方式被透明地导入或导出，它还具有很高的可用性，支持链路级和应用级容错功能。域网关可以把事务和安全上下文传递到其他域，从而保证了跨域交易的完整性和安全性。Tuxedo 的域网关也存在一些不足，它目前只支持“请求/应答”（同步、异步、嵌套、转发）、会话、队列 3 类通信模式，还不

支持跨域消息通告和事件订阅方式。

多机与多域相比，各有优缺点。多机方式成本低，但可管理性差，缺乏弹性；多域模式易于管理和维护，适应性强，但成本高。一般来说，跨越多个地域、组织结构复杂、业务相对独立的分布式系统，使用多域模式比较合适；而因为单机处理能力有限，需要使用集群方式来加强计算能力和可靠性的系统，使用多机模式比较合适。

多机模式与多域模式存在很多相似的地方，大多数多机应用程序可以改成多域模式，并不会造成性能损失，反之亦然。但是在有的情况下，使用多机模式不但可以充分利用硬件资源，而且可以节省成本，但是系统却缺乏弹性。使用多域模式虽然成本比较高，但是系统的可靠性和适应性却比较好。下面从管理模式、适应性和成本 3 个方面来作比较。

（1）管理模式：多机应用采用的是集中式管理模式，所有管理工作都集中在 Master 节点上来完成，包括应用程序的配置、启动、关闭、运营维护和日常事务管理等。应用程序启动时，Master 节点会把配置文件依次传递到每一个节点，然后通过 tlisten 进程启动成员节点上的 Tuxedo 系统，关闭过程也是一样。主机之间除了要传递配置文件和管理指令外，还需要交换大量的业务数据，这就对网络的速度和响应性提出了较高的要求，所有节点最好都处在同一个高速局域网中，否则很容易造成节点被隔离。如果构成多机的节点比较多，那么应用程序的启动和关闭都将是一个非常耗时的过程，所以建立节点数量不要超过 5 个。

多域应用采用的是分散式的管理模式，即每个域都是一个独立的管理单元，都有自己的配置文件，它们通过网关连在一起，协同处理客户请求。域和域之间完全是松耦合的，它们都可以独立地启动和关闭，它们只通过网关交换业务数据，因此对网络的响应性要求不高。

（2）适应性：虽然多机和多域应用都支持数据依赖路由、负载均衡和容错，但是它们的适应性是不同的。多机模式的适应性较差，它很难满足业务模块的变动和动态增长的需求，系统一旦启动，很难再动态添加新的节点和业务模块，因此仅适合于构建业务成熟、模块相对固定的应用。相比之下，多域模式的适应性要好一些，很容易满足业务变动和模块动态增长的需求。随着业务的发展，企业可能会使用 Tuxedo 来开发更多的系统，这些新系统可以很容易地使用域网关实现和现有系统的互联，实现业务渗透。如果某个子系统或模块需要升级，只需要把它所在的域网关关闭即可，而不会对其他域造成不良影响。由此可见，对于不断发展变化的企业业务来说，多域模式更有弹性。

（3）成本：在多机应用中，只有 Master 节点需要安装 License，其他节点均不需要安装，Tuxedo 系统会根据配置在节点之间分配 License。例如，用户购买了 250 个 Tier4 的并发 License，要在 A、B、C 3 个节点之间分配，那么可以给 A 分配 90 个，给 B 和 C 各分配 80 个，也可以按其他比例来灵活分配。

在多域应用中，每个域都必须单独安装有效的 License。例如，某个用户有 A、B、C 3 个域，每个域都按照 Tier 3 来计算 License，那么该用户要购买的并发 License 数量为 300 个，比多机模式多了 50 个，无形中就增加了用户对 IT 系统的总体拥有成本。

另外，每个域的管理程序也会占用并发数，域的数量越多，占用的也就越多，因此成本也就越高。

7.5　Tuxedo 与多种平台连通

7.5.1　与其他系统的互连概要

（1）通过 TDOMAIN 与其他的 Tuxedo 应用系统互连，它已经在 Tuxedo 的软件包中，安装完 Tuxedo Server 端就有该功能。

（2）通过 TMA 与大机（MainFrame）上的系统互连。

（3）通过 BEA JOLT 产品与 JAVA 客户端互连，一般是与 JAVA 应用服务器互连，如 WebLogic、WEBSPERE、IPLANTET 等。

（4）通过 WTC 与 WebLogic 互连（WebLogic 6.0 之后的版本才支持）。

（5）通过 JCA 与 JAVA 应用服务器互连。

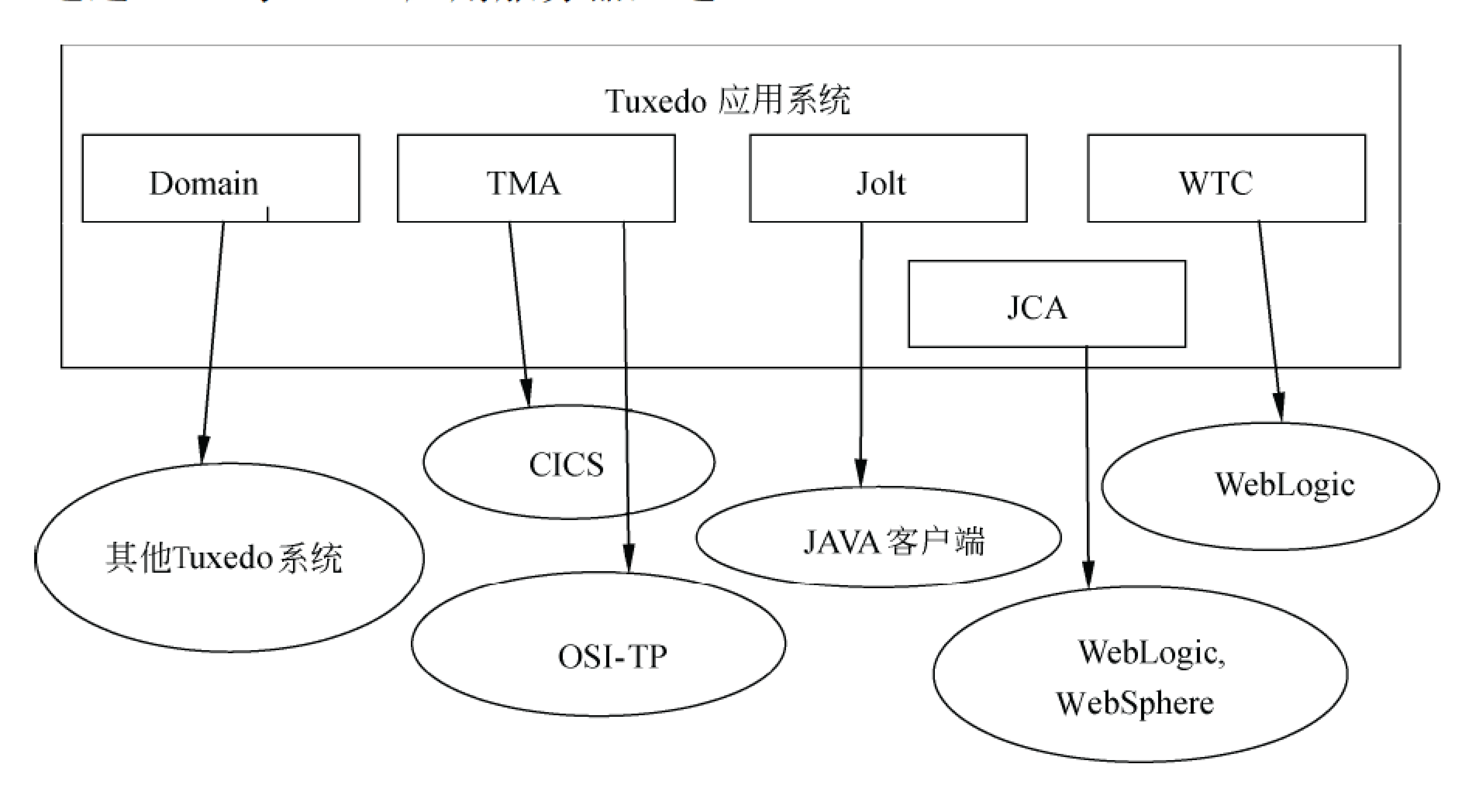


图 7-5

7.5.2　经典的 WTC

域网关不仅可以连接多套 Tuxedo 应用，还可以实现 Tuxedo 应用与其他平台的连接。例如与 WebLogic 应用的连接，就是通过在 Tuxedo 应用中配置域网关，在 WebLogic 中使用 WTC（WebLogic Tuxedo Connector）实现的。而与大机遗留系统的连接，也是基于域网关，通过 TMA（Tuxedo Mainframe Adapter）进行的。

Tuxedo 和 WebLogic 作为最优秀的中间件产品，在银行、电信、金融等行业广泛使用，通常采用 Tuxedo 实现系统的核心业务，用 WebLogic 作为系统扩展到 Internet 的平台，实现电子商务，由 WebLogic 调用 Tuxedo 上的服务，所以 Tuxedo 与 WebLogic 之间的互连经常遇到。

WebLogic 与 Tuxedo 的互连有两种方式，通过 JOLT 或通过 WTC(WebLogic Tuxedo CONNECTOR)。WTC 不仅能让 WebLogic 调用 Tuxedo 中的 SERVICE，而且能让 Tuxedo

调用 WebLogic 中的 EJB；而 JOLT 只能让 WebLogic 调用 Tuxedo。但 JOLT 可以使 Tuxedo 与 WEBSPERE 等其他应用服务器相连，而 WTC 只能用于 WebLogic 与 Tuxedo 之间进行互连。

对于 WTC 需要配置如下信息，见表 7-3。

表 7-3

类型	描述
DM_RESOURCES	指定总体域配置版本信息。本节中的唯一参数是 Version=string，其中 string 是一个字段，用户可以输入当前 DMCONFIG 文件的版本号，这个字段不由软件检查
WTCServer	包含在 WebLogic Server 和 Tuxedo 之间进行连接时所需的互操作性特性的父 Mbean。在使用管理控制台进行配置时定义 WTC 服务
WTCLocalTuxDom	提供配置信息以便将可用的远程 Tuxedo 域连接到 WTC 服务，用户必须至少配置一个 Tuxedo 访问点，在使用管理控制台进行配置时定义本地 Tuxedo 访问点
WTCRemoteTuxDom	提供配置信息以便将 WTC 服务连接到可用的远程 Tuxedo 域，可以配置多个远程域，在使用管理控制台进行配置时定义远程 Tuxedo 访问点
WTCExport	提供由本地 Tuxedo 访问点导出的服务的相关信息。在使用管理控制台进行配置时定义导出服务
WTCImport	提供有关在远程域上导入并可用的服务的相关信息。在使用管理控制台进行配置时定义导入服务
WTCResources	指定域的全局字段表类、视图表类及应用程序密码。在使用管理控制台进行配置时定义资源

Tuxedo 要做的配置如下。

（1）修改 D:\SIMPAPP\SETENV.CMD。

示例 7-3：

```
set TUXDIR=d:\tuxedo65
set WSNADDR=//DEMOSERVER:8888
set APPDIR=d:\simpapp
set PATH=%TUXDIR%\bin;%APPDIR%;%PATH%
set TUXCONFIG=%APPDIR%\tuxconfig
set BDMCONFIG=%APPDIR%/dbmconfig
```

（2）修改 D:\SIMPAPP\UBBDOMAIN。

示例 7-4：

```
*RESOURCES
IPCKEY 123456
DOMAINID simpapp
MASTER simple
MAXACCESSERS 10
MAXSERVERS 5
MAXSERVICES 10
```

```
MODEL SHM
LDBAL N

*MACHINES
DEMOSERVER LMID=simple
APPDIR="d:\simpapp"
TUXCONFIG="d:\simpapp\tuxconfig"
TUXDIR="d:\tuxedo65"

*GROUPS
GROUP1
LMID=simple GRPNO=1 OPENINFO=NONE
GROUP2
LMID=simple GRPNO=2 OPENINFO=NONE
*SERVERS
DEFAULT:
CLOPT="-A"
simpserv SRVGRP=GROUP1 SRVID=1
DMADM SRVGRP=GROUP2 SRVID=1
GWADM SRVGRP=GROUP2 SRVID=2
GWTDOMAIN SRVGRP=GROUP2 SRVID=3
*SERVICES
TOUPPER
```

（3）修改 D:\SIMPAPP\DOM1CONFIG。

修改 dom1config，加入 TLOG 的路径和 AUDITLOG 的路径。然后修改 TDOM1 的 NWADDR="IP:PORT"为 Tuxedo 的 IP 和端口，TDOM2 的 NWADDR="IP:PORT"为 WebLogic 的 IP 和端口。

示例 7-5：

```
*DM_RESOURCES
VERSION=U22

*DM_LOCAL_DOMAINS
TDOM1 GWGRP=GROUP2
TYPE=TDOMAIN
DOMAINID="TDOM1"
BLOCKTIME=20
MAXDATALEN=56
MAXRDOM=89
TLOG="d:\simpapp\TLOG"
AUDITLOG="d:\simpapp\AUDITLOG"

*DM_REMOTE_DOMAINS
TDOM2 TYPE=TDOMAIN
```

```
DOMAINID="TDOM2"

*DM_TDOMAIN
TDOM1 NWADDR="//DEMOSERVER:9998"
TDOM2 NWADDR="//DEMOSERVER:9999"

*DM_REMOTE_SERVICES
TOLOWER RDOM="TDOM2"
```

（4）tmloadcf -y ubbdomain。

（5）dmloadcf -y dom1config。

（6）buildserver -o simpserv -f simpserv.c -s TOUPPER。

（7）将 examples/wtc/atmi/simpapp/simpserv 下的 tolower.c 拷贝到 D:\SIMPAPP，并执行 buildclient -v -f tolower.c -o tolower。

WebLogic 要做的配置如下。

（1）修改 WebLogic 目录下 config/examples/setExamplesEnv.cmd 及 startExamplesServer.cmd，在 CLASSPATH 中加入 d:\weblogic\wtc1.0\lib/jatmi.jar，执行 setExamplesEnv.cmd。

（2）修改 bdmconfig.xml。

修改<!DOCTYPE>中的 [WTC installation directory]\weblogic\wtc\gwt\wtc_ config_1_0.dtd(WLS60)，修改其中 TDOM1 和 TDOM2 的 NWADDR，确保和 Tuxedo 中的 dom1config 的定义一致。

（3）修改后的 bdmconfig.xml 内容如下。

示例 7-6：

```
<?xml version="1.0"?>
<!DOCTYPE BDMCONFIG SYSTEM
"file:D:\weblogic\wtc1.0\weblogic\wtc\gwt\wtc config 1 0.dtd">
<!--Java and XML-->
<WTC CONFIG>
<BDMCONFIG>
<T_DM_LOCAL_TDOMAIN AccessPoint="TDOM2">
<WlsClusterName>Coolio</WlsClusterName>
<AccessPointId>TDOM2</AccessPointId>
<Type>TDOMAIN</Type>
<Security>NONE</Security>
<ConnectionPolicy>ON DEMAND</ConnectionPolicy>
<BlockTime>30</BlockTime>
<NWAddr>//DEMOSERVER:9999</NWAddr>
<!-- Example address: //mydomain.acme.com:9999 -->
<Interoperate>Yes</Interoperate>
</T DM LOCAL TDOMAIN>
<T DM REMOTE TDOMAIN AccessPoint="TDOM1">
<LocalAccessPoint>TDOM2</LocalAccessPoint>
<AccessPointId>TDOM1</AccessPointId>
```

```
<Type>TDOMAIN</Type>
<NWAddr>//DEMOSERVER:9998</NWAddr>
<!-- Example address: //mydomain.acme.com:9998 -->
</T_DM_REMOTE_TDOMAIN>
<T_DM_EXPORT ResourceName="TOLOWER" LocalAccessPoint="TDOM2">
    <EJBName>tuxedo.services.TOLOWERHome</EJBName>
</T_DM_EXPORT>
<T_DM_IMPORT ResourceName="TOUPPER"  LocalAccessPoint="TDOM2"
    RemoteAccessPointList="TDOM1">
<TranTime>600</TranTime>
</T_DM_IMPORT>
</BDMCONFIG>
</WTC_CONFIG>
```

（4）执行 java weblogic.wtc.gwt.WTCValidateCF bdmconfig.xml，监测其是否正确。

（5）在 D:\SIMPAPP 下执行 build 命令，然后在 D:\weblogic\wtc1.0\examples\simpserv 下执行 build 命令。

（6）启动 WebLogic。

（7）察看 wtc_tolower.jar、wtc_toupper.jar 是否 deploy 成功。

（8）建立一个 WebLogic StartUp Class。

classname 为 weblogic.wtc.gwt.WTCStartup，参数为 BDMCONFIG=D:\SIMPAPP\bdmconfig.xml，并设置其 TARGET 为 EXAMPLESERVER。

（9）建立一个 WebLogic ShutDown Class。

classname 为 weblogic.wtc.gwt.WTCShutdown，并设置其 TARGET 为 EXAMPLESERVER。

（10）重新启动 WebLogic，并注意 WebLogic 的启动日志，看 StartUp Class 启动过程是否成功，失败会报错，成功没有提示信息。

如果失败，没有检查 config.xml 中是否为：

```
<StartupClass
    Arguments="BDMCONFIG=d:\wtc load4\examples\simpapp\bdmconfig.xml"
    ClassName="weblogic.wtc1.0.gwt.WTCStartup" FailureIsFatal="false"
    Name="MyWTCStartup Class" Targets="myserver"
/>
<ShutdownClass Arguments="" ClassName="weblogic.wtc1.0.gwt.WTCShutdown"
    Name="MyWTCShutdown Class"
/>
```

（11）启动 Tuxedo 运行例子。

在 d:/simpapp 下执行 run.cmd，这是 WebLogic 作 CLIENT 端调用 Toupper ejb，由 Toupper ejb 调用 Tuxedo 的 SERVICE：TOUPPER。

在 d:/simpapp 下执行 run.cmd，这是 Tuxedo 作 CLIENT 端调用 WebLogic 中的 Tolower ejb。

7.5.3 JCA Adapter 新特性

1. JCA 的基础知识

Oracle Tuxedo JCA 适配器是一个基于 JCA 的适配器，它提供遵从 JCA 1.5 的应用服务器和 Oracle Tuxedo 服务系统之间的双向服务调用。JCA 适配器与 JCA 交易标准一样支持 Tuxedo 局部和全局事务。它支持连接管理、交易传播、身份传递以及连接安全。连接安全使用 SSL/TLS 的行业标准或一个专用的高效算法来实现。

Oracle Tuxedo JCA 适配器支持 Tuxedo 同步和异步调用。它支持 JCA 标准常见的用户界面（CCI）以及用作访问 Oracle Tuxedo 服务的 JEE 应用程序（EJBs、POJOs、Servlets/ JSPs 等）的 Oracle Tuxedo 标准 JATMI（Java Application-To-Monitor）。标准的 ATMI 被用在从 Oracle Tuxedo 应用程序访问 JEE 应用程序服务。

Oracle Tuxedo JCA 适配器可以配置在 JCA 标准资源适配器配置文件 ra.xml 和 Oracle Tuxedo 配置文件 dmconfig.xml 中。此外，Tuxedo JCA 适配器可以用本地应用服务器的方法部署配置。

图 7-6 为 Tuxedo JCA 适配器的结构图。

- Oracle Tuxedo JCA Adaptor

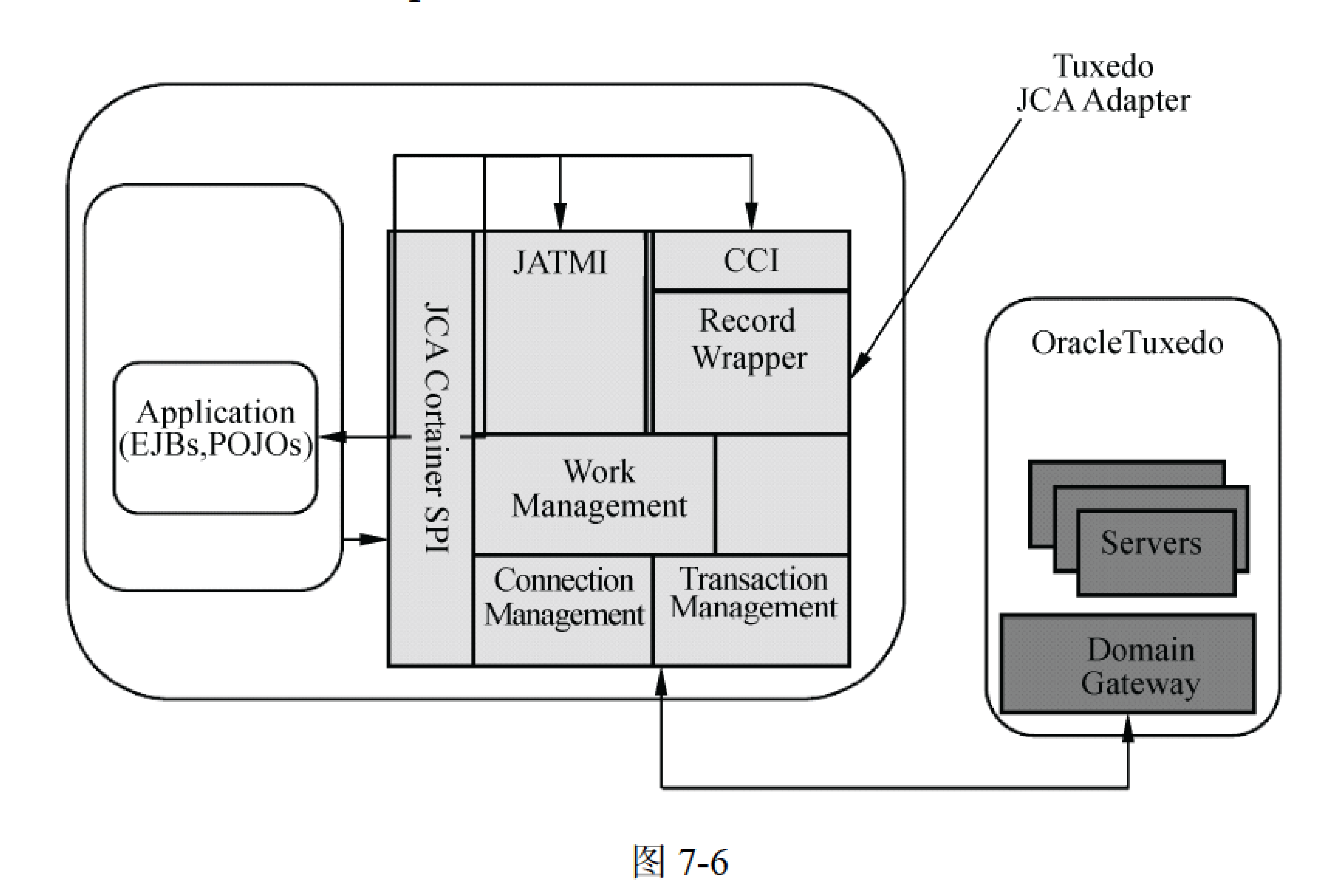


图 7-6

2. 运用 JCA 实现 Tuxedo 与 WebLogic 互联

一般通过 JCA 实现 Tuxedo 和 WebLogic 互联，所需的环境如下。

（1）WebLogic10gR3 或更高版本。

（2）Tuxedo 10gR3 或更高版本。

（3）确认 C 编译器已安装好。

（4）ant 1.6.5 或更高版本。

（5）把 ant-contrib-1.0b3.jar 放到 CLASSPATH 下。

（6）JDK 1.5/1.6。

（7）Tuxedo JCA 适配器 com.oracle.tuxedo.TuxedoAdapter.rar。

具体操作测试步骤一般如下。

（1）复制 com.oracle.tuxedo.TuxedoAdapter.rar 文件到当前工作目录。

（2）使用“jar”命令 un-jar 这个文件。之后会创建如下目录。

```
./src             存放所有源代码
./setenv.cmd      在 Windows 下设置环境变量的脚本
./setenv.sh       在 UNIX 下设置环境变量的脚本
./readme.txt      关于这个例子的解释
./build.xml       "ant"脚本
```

（3）根据用户自己的环境修改 setenv.cmd 或者 setenv.sh 设置好环境变量。上面提到的这两个脚本必须运行在$WORKDIR 下，$WORKDIR 是存放 build.xml 文件的目录。

（4）在当前工作目录下创建一个新的目录“jdom”。

（5）在刚创建好的$WORKDIR/jdom 目录下配置一个新的 WebLogic 域。

（6）运行“ant setup”。执行此步骤会创建一个 Tuxedo 应用程序的例子，包括创建 Tuxedo 应用程序服务器。

（7）运行“ant tja.config”，会创建 Tuxedo JCA 适配器配置文件。这个配置文件会以“bdmconfig.xml”的名字存放在$WORKDIR/adapter 目录下。

（8）使用 startWebLogic.cmd 或者 startWebLogic.sh 启动 WebLogic。

（9）运行“ant rar”，这会为 Tuxedo JCA 适配器创建 RAR 文件，并把它复制到 WebLogic 域下的 autodeploy 文件夹下。

（10）运行“ant ejb”，会通过 Tuxedo JCA 适配器创建一个 EJB 作为 Tuxedo 客户端。这个 EJB 也会被复制到 WebLogic 域下的 autodeploy 目录下。在进入下一步之前等待部署完成。

（11）运行“ant client.str”。此步将有一个 Java 客户端调用 Tuxedo TOUPPER 服务。

（12）运行“ant shutdown”。关闭 Tuxedo 应用程序。

（13）关闭 WebLogic。

更多详细内容请参考官方文档：

http://download.oracle.com/docs/cd/E15261_01/jca/docs11gr1/index.html

第 8 章　Tuxedo 常用的管理操作

8.1　启停 Tuxedo 应用

8.1.1　相关应用环境

1. tux.env 环境文件

示例 8-1：

```
TUXDIR=/home/landingbj/tuxedo/tuxedo11gR1; export TUXDIR
JAVA_HOME=$TUXDIR/jre; export JAVA_HOME
JVMLIBS=$JAVA HOME/lib/i386/server:$JAVA HOME/jre/bin
PATH=$TUXDIR/bin:$JAVA HOME/bin:$PATH; export PATH
COBCPY=:$TUXDIR/cobinclude; export COBCPY
COBOPT="-C ANS85 -C ALIGN=8 -C NOIBMCOMP -C TRUNC=ANSI -C OSEXT=cbl";
export COBOPT
SHLIB PATH=$TUXDIR/lib:$JVMLIBS:$SHLIB PATH; export SHLIB PATH
LIBPATH=$TUXDIR/lib:$JVMLIBS:$LIBPATH; export LIBPATH
LD_LIBRARY_PATH=$TUXDIR/lib:$JVMLIBS:$LD_LIBRARY_PATH;
export LD LIBRARY PATH
WEBJAVADIR=$TUXDIR/udataobj/webgui/java; export WEBJAVADIR
LANG=C; export LANG
APPDIR=/home/landingbj/test; export APPDIR
TUXCONFIG=$APPDIR/tuxconfig; export TUXCONFIG
```

安装 Tuxedo 时会自动生成一个环境文件，其中包括 Tuxedo 安装目录、Java 安装目录、PATH、LIB 路径（不同的操作系统对应不同的变量名：LD_LIBRARY_PATH 用在 solaris 或 Linux 上，SHLIB_PATH 用在 HP-UX 上，LIBPATH 在 IBM AIX 上使用）；

还有一些是应用配置需要的，如字符集、应用目录、domain 配置文件位置等。

2. UBBCONFIG 配置文件

示例 8-2：

```
#   (c) 2003 BEA Systems, Inc. All Rights Reserved.
#ident  "@(#) samples/atmi/simpapp/ubbsimple   $Revision: 1.7 $"

#Skeleton UBBCONFIG file for the Tuxedo Simple Application.
```

```
#Replace the        <bracketed> items with the appropriate values.

*RESOURCES
#IPCKEY             <Replace with a valid IPC Key>

#Example:
IPCKEY              123456

DOMAINID            simpapp
MASTER              simple
MAXACCESSERS        11
MAXSERVERS          5
MAXSERVICES         10
MODEL               SHM
LDBAL               N

*MACHINES
DEFAULT:
        APPDIR="/home/landingbj/simpapp"
        TUXCONFIG="/home/landingbj/simpapp/tuxconfig"
        TUXDIR="/home/landingbj/tuxedo/tuxedo11gR1"
        MAXWSCLIENTS = 10
#Example:
#       APPDIR="/home/me/simpapp"
#       TUXCONFIG="/home/me/simpapp/tuxconfig"
#       TUXDIR="/usr/tuxedo"

landingbj   LMID=simple

#Example:
#beatux LMID=simple

*GROUPS
GROUP1
    LMID=simple GRPNO=1 OPENINFO=NONE

GROUP2
    LMID=simple GRPNO=2 OPENINFO=NONE

*SERVERS
DEFAULT:
        CLOPT="-A"

simpserv    SRVGRP=GROUP1 SRVID=1
```

```
DMADM        SRVGRP=GROUP2 SRVID=1
GWADM        SRVGRP=GROUP2 SRVID=2
GWTDOMAIN    SRVGRP=GROUP2 SRVID=3
WSL SRVGRP=GROUP1 SRVID=2 CLOPT="-A -t -- -n //192.168.0.34:8888
-m 2 -M 5 -x 10"
*SERVICES
TOUPPER
```

运行 TOUPPER 服务需要配置的几个关键点如下。

- **IPCKEY** 对应 IPC 资源的键值，每个在同一主机上的配置文件的 IPCKEY 必须保证不同。
- **DOMAINID** 域 ID。
- **APPDIR** 应用目录。
- **TUXCONFIG** 域配置文件路径。
- **TUXDIR** Tuxedo 安装路径。
- **LMID** 机器逻辑名。
- **SERVICES** 用来定义服务相关属性。

8.1.2 启动 Tuxedo 应用

tmboot 常用参数如下。

- **–A** 启动所有管理 SERVER。
- **–l** 启动指定逻辑机器的所有 SERVER。
- **–g** 启动某一组的 SERVER。
- **–i** 启动指定 SERVER id 的 SERVER。
- **–s** 启动指定可执行文件名的 SERVER。
- **–w** 快速启动。
- **–y** 启动时不再提示是否确定启动。
- **–e** 指定一个命令，如果任何一个 SERVER 启动失败就执行这个命令。

8.1.3 停止 Tuxedo 应用

tmshutdown 常用参数如下。

- **–A** 关闭所有管理 SERVER。
- **–l** 关闭指定逻辑机器的所有 SERVER。
- **–g** 关闭某一组的 SERVER。
- **–i** 关闭指定 SERVER id 的 SERVER。
- **–s** 关闭指定可执行文件名的 SERVER。
- **–y** 关闭时不再提示是否确定关闭。
- **–e** 指定一个命令，如果任何一个 SERVER 关闭失败就执行这个命令。

- **–w**　<delay>等待一段时间，然后执行强制关闭 SERVER。
- **–c**　关闭 SERVER 时不管客户端是否有连接。

8.2　管理和监控

8.2.1　一般管理监控 tmadmin

1．tmadmin 的主要功能

tmadmin 通常的管理监控工作有以下 3 类。

（1）查看系统运行状态，一般查看以下信息。

- -应用
- -服务
- -客户端
- -交易
- -队列
- -组
- 会话
- 网络

（2）动态修改服务或 Tuxedo 系统参数。

（3）进行启动、关闭、迁移服务进程等管理任务。

2．tmadmin 的常用操作

（1）输出服务进程信息 printserver（简写为 psr）。

示例 8-3：

```
Prog Name    Queue Name   Grp Name     ID RqDone Load Done Current Service
---------    ----------   --------     -- ------ --------- ---------------
BBL          133333        simple      0    0        0      (  IDLE )
DMADM         00002.00001  DWGRP       1    12       600    (  IDLE )
simpserv     00001.00001  GROUP1      1    0        0      (  IDLE )
GWADM         00032.00020  GWGRP2      20   0        0      (  IDLE )
GWADM         00031.00020  GWGRP       20   0        0      (  IDLE )
GWTDOMAIN    gw2          GWGRP2      30   0        0      (  IDLE )
GWTDOMAIN    gw1          GWGRP       30   0        0      (  IDLE )
```

列号及其描述如下。

第 1 列：服务进程的可执行文件名。

第 2 列：服务进程连接的队列名。

第 3 列：服务进程所在的组名。

第 4 列：服务进程的 id。
第 5 列：服务进程已经处理的请求数。
第 6 列：服务进程处理的全部请求的 LOAD。
第 7 列：服务进程正在处理的服务，若为 IDLE 则服务进程当前是空闲的。
（2）输出服务信息 printservice（简写为 psc）。
示例 8-4：

```
Service Name Routine Name Prog Name  Grp Name  ID  Machine  # Done Status
------------ ------------ ---------  --------  --  -------  ------ ------
DMADMIN      DMADMIN         DMADM     DWGRP    1  simple     0    AVAIL
TOUPPER      TOUPPER      simpserv    GROUP1    1  simple     0    AVAIL
TDOM12              GWS      GWADM    GWGRP2   20  simple     0    AVAIL
TDOM1               GWS      GWADM     GWGRP   20  simple     0    AVAIL
TOLOWER             GWS  GWTDOMAIN    GWGRP2   30  simple     0    AVAIL
TMS                 GWS  GWTDOMAIN    GWGRP2   30  simple     0    AVAIL
TOLOWER             GWS  GWTDOMAIN     GWGRP   30  simple     0    AVAIL
TMS                 GWS  GWTDOMAIN     GWGRP   30  simple     0    AVAIL
```

列号及其描述如下。
第 1 列：服务名。
第 2 列：服务函数名。
第 3 列：提供服务的程序名。
第 4 列：服务所在的组名。
第 5 列：服务所在的服务进程的 id。
第 6 列：提供服务的机器的 LMID。
第 7 列：服务已经执行的次数。
第 8 列：服务当前的状态。
（3）输出服务进程的队列信息 printqueue（简写为 pq）。
示例 8-5：

```
Prog Name      Queue Name   # Serve  Wk Queued  # Queued  Ave. Len   Machine
---------     -----------   -----    ---------  ----      -----      -------
GWADM         00032.00020       1    -                0        -      simple
BBL                133333       1    -                0        -      simple
DMADM         00002.00001       1    -                0        -      simple
GWADM         00031.00020       1    -                0        -      simple
GWTDOMAIN             gw2       1    -                0        -      simple
Simpserv      00001.00001       1    -                0        -      simple
GWTDOMAIN             gw1       1    -                0        -      simple
```

列号及其描述如下。
第 1 列：队列连接的服务进程名。

第 2 列：队列名，由 RQADDR 参数指定或 Tuxedo 生成。
第 3 列：连接的服务进程数。
第 4 列：当前队列的所有请求的 LOAD。
第 5 列：当前队列中的请求数。
第 6 列：平均队列长度。
第 7 列：队列所在机器的 LMID。
（4）输出客户端信息 printclient（简写为 pclt）。
示例 8-6：

```
LMID          User Name        Client Name     Time     Status     Bgn/Cmmt/Abrt
---------      ---------       -------------   ------   -------    -------------
simple         landingbj          tmadmin       0:17:39 IDLE       0/0/0
```

列号及其描述如下。
第 1 列：已经注册的客户端所在机器的 LMID。
第 2 列：用户名，由 tpinit()提供的。
第 3 列：客户端名，由 tpinit()提供的。
第 4 列：客户端连接后经过的时间。
第 5 列：客户端状态。

- **IDLE**　表示客户端目前没有任何未完成的服务请求或会话。
- **IDLET**　表示客户端启动了一个全局事务，目前空闲。
- **BUSY**　表示客户端的服务请求或会话正在被处理。
- **BUSYT**　表示客户端处于全局事务中的服务请求或会话正在被处理。

（5）启动/提交/回滚的事务数。
① 输出公告板（BB）统计信息 bbstats（简写为 bbs）。
示例 8-7：

```
Current Bulletin Board Status:
Current number of servers: 7
Current number of services: 20
Current number of request queues: 7
Current number of server groups: 5
Current number of interfaces: 0
```

② 查看节点间通信状态 printnet（简写为 pnw）。
示例 8-8：

```
> pnw SITE12
SITE12  Connected To:  msgs sent    msgs received
        SITE14         61904         62319
        SITE13         61890         62288
        SITE11         15972         13564
```

③ default
设置默认管理方案。

示例 8-9：

```
default -m site1
```

表示接下来管理时只输出逻辑机器 site1 的信息。

① quit（简写为 q）

退出管理界面

② help

显示帮助

8.2.2 域管理监控 dmadmin

1．知识回顾

（1）域配置文件

文本文件：tuxedo.dm；二进制文件：bdmconfig；环境变量：BDMCONFIG。

$dmloadcf tuxedo.dm //生成二进制文件。

$dmunloadcf >tuxedo.dm.bak //反向生成文本文件。

（2）域网关服务器

GWTDOMAIN 等服务器进程。

（3）域管理服务器

网关管理（启动时在 DMADM 进行注册，并定期向其汇报自己状态）：GWADM。

域管理服务器（提供 GWADM 注册服务）：DMADM。

（4）域管理工具

dmloadcf：生成二进制配置文件。运行时检查$TUXDIR/udataobj/DMTYPE 文件，查看域类型是否正确。

Dmunloadcf：将当前域使用的配置导出到文本文件。

dmadmin：域管理监控工具。

2．dmadmin 常用操作

（1）查看域连接状态 printdomain（简写为 pd）

示例 8-10：

```
$dmadmin
dmadmin - Copyright (c) 1996-1999 BEA Systems, Inc.
Portions * Copyright 1986-1997 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
Tuxedo is a registered trademark.
> pd -d LDOM0 //显示域连接状态
Local domain :LDOM0
```

```
Connected domains:
Domainid: trade02

Disconnected domains being retried:
Domainid: trade03
```

（2）手工连接一个域连接 connect（简写为 co）

示例 8-11：

```
> co -d LDOM0 -R RDOM1
Operation completed successfully. Use printdomain(pd) to obtain results.
```

（3）手工断开一个域连接 disconnect（简写为 dco）

示例 8-12：

```
> dco -d LDOM0 -R RDOM1
Operation completed successfully. Use printdomain(pd) to obtain results.
```

8.2.3　队列管理监控 qmadmin

1．知识回顾

/Q 是 Tuxedo 系统的一个重要组成部分，它提供了一种可靠队列机制，允许消息按某种排队规则存储到磁盘上或内存中，然后再转发给其他进程。这种存储转发机制可以保证在两个通信实体之间传递的消息不丢失、不重传，从而保证交易的完整性和可靠性。

Tuxedo /Q 提供管理工具和编程接口用于对/Q 进行管理和操作。

（1）/Q 的组成（图 8-1）

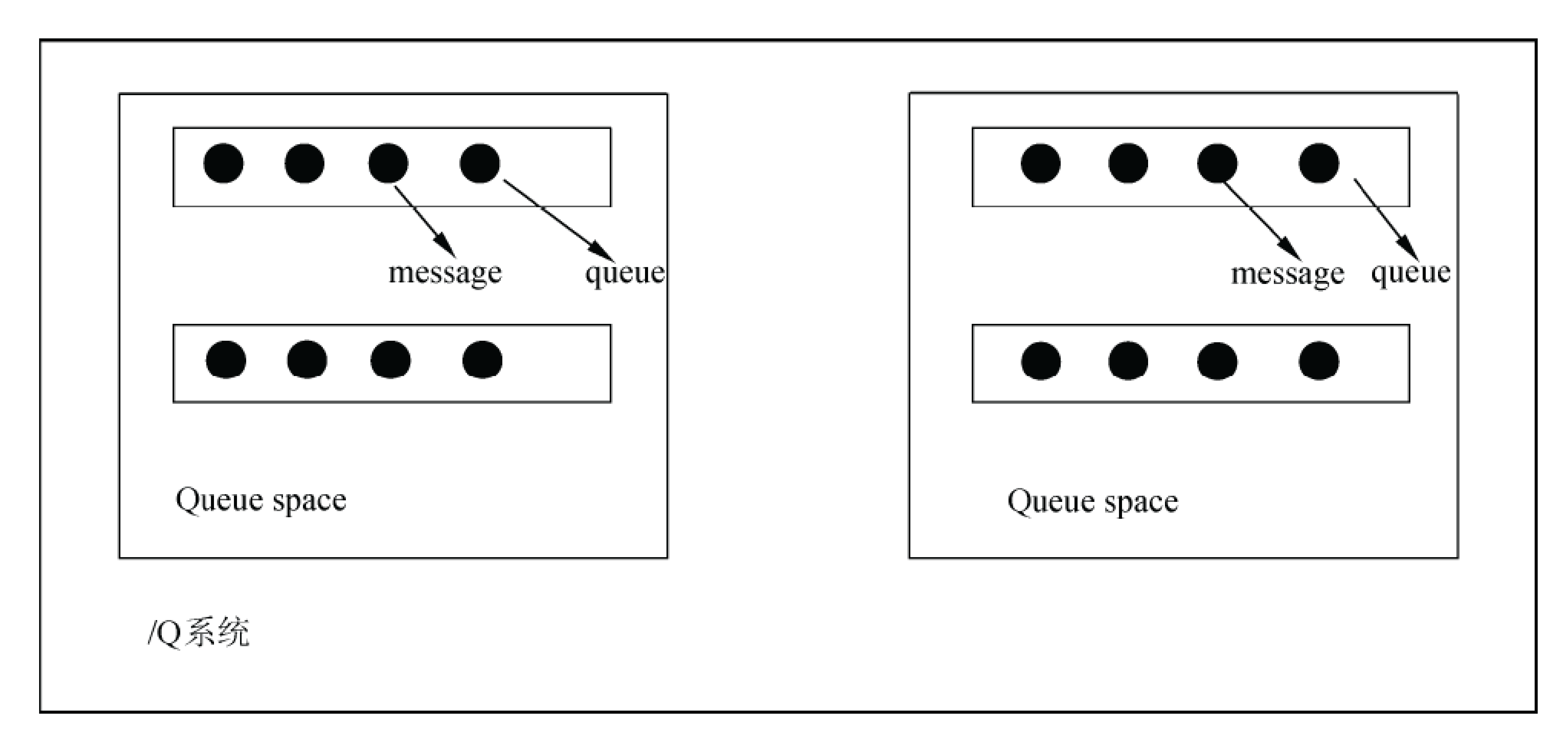


图 8-1

（2）/Q 的使用方式（图 8-2）

图中左侧为 Tuxedo 客户端，右侧为 Tuxedo 服务进程，中间为 Tuxedo 可靠消息队列系统/Q。右侧服务进程提供了两个服务：SERVICE1 和 SERVICE2。在消息队列系统/Q 中，

一个 QUEUE SPACE 对应一个 GROUP，TMS_QM 是/Q 的事务管理进程，在该 GROUP 中要进行定义，QUEUE SPACE 中定义了 4 个消息队列，SERVICE1 和 SERVICE2 分别对应于同名的两个服务（SERVICE）的队列（这是一种命名规则，客户机若请求服务器中的 SERVICE1 服务，就把请求消息放入 SERVICE1 队列中）。SERVICE1 的处理结果放到 CLIENT_REPLY1 中，如果 SERVICE1、SERVICE2 在处理过程中发生错误，把错误信息保存到队列 FAILUREQ 中。

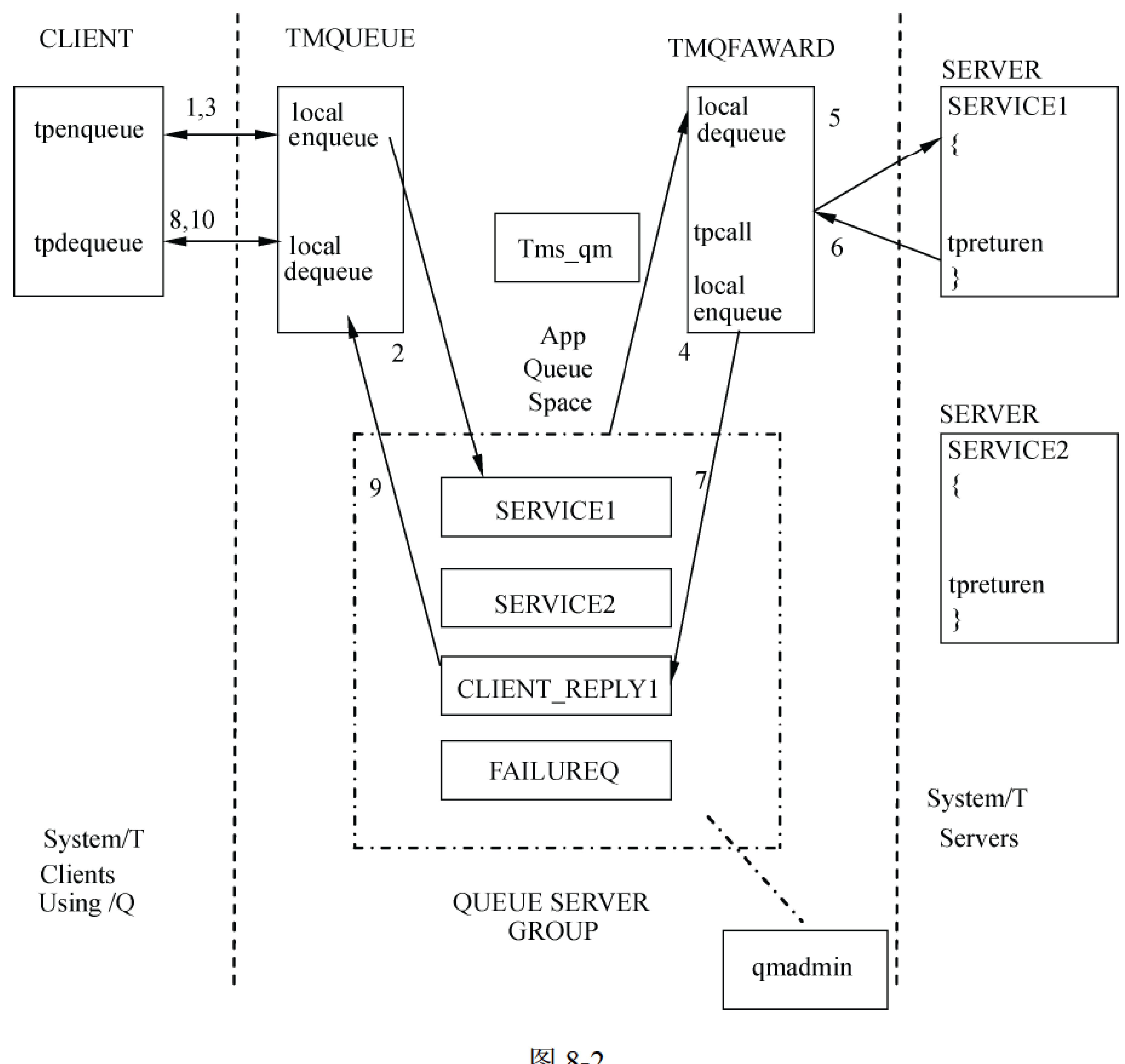


图 8-2

/Q 有两种使用方式，称之为基本模式和转发模式。

（1）基本模式

只用到 TMQUEUE，不使用 TMQFORWARD，不调用 SERVICE 对 QUEUE 中的消息进行处理。

流程说明如下（在上面的例子中，不需要进行第 4-7 中的操作，也不需要定义与 SERVICE 同名的 QUEUE）。

① 客户端调用 tpenqueue()把数据发送到 SERVICE1 队列。

② TMQUEUE 接收 tpenqueue()发送来的数据，并把它们保存到 SERVICE1 队列中。

③ 如果以上操作成功，那么 tpenqueue()返回成功。

④ 客户端调用 tpdequeue()，请求从 SERVICE1 队列中取数据。

⑤ TMQUEUE 收到该请求，它把相应的消息从 SERVICE1 队列中取出，并发送给客

户端。

（2）转发模式

用到 TMQUEUE 和 TMQFORWARD，要定义与 SERVICE 同名的 QUEUE，并调用 SERVICE 对 QUEUE 中的消息进行处理。

流程说明如下。

① 客户端调用 tpenqueue()把消息发送到 SERVICE1 队列。

② TMQUEUE 接收 tpenqueue()发送来的消息，并把数据保存到 SERVICE1 队列中。

③ 如果以上操作成功，那么 tpenqueue()返回成功。

④ TMQFORWARD 在设定的周期，从 SERVICE1 队列中取消息。

⑤ TMQFORWARD 开始一个全局事务，并调用 SERVICE1 服务来处理这个消息。

⑥ SERVICE1 服务把处理的结果用 tprerurn()返回给 TPQFORWARD。

⑦ TPQFORWARD 把收到的处理结果发送到 REPLYQ，这里是 CLIENT_REPLY1。

⑧ 客户端调用 tpdequeue()，请求从 REPLYQ 中取回响应消息。

⑨ TMQUEUE 收到该请求，它把相应的消息从 CLIENT_REPLY1 中取出，并发送给客户端。

⑩ 如果以上操作成功，那么 tpdequeue()返回成功。

注 意

当一条消息从 QUEUE 中被取出后，在该 QUEUE 中它将被删除，其他的进程就看不到这个消息了。如果有多个进程同时要取这个消息，只有最早的那个进程能取到该消息。

（3）/Q 的使用场合

首先，/Q 的常见用法是用于实现数据的可靠传送，把数据从一台机器可靠地传送到另一台机器，例如，在电信计费业务中，可以用/Q 把采集到的计费数据发送到计费中心进行处理；在银行中，不同的银行间可用/Q 传送结算数据。数据传送可以是在 Tuxedo 客户端与 Tuxedo 服务端之间，或 Tuxedo 服务端与服务端之间。

其次，可以用/Q 实现工作流，如图 8-3 所示，前面的处理流程把处理结果保存到 QUEUE 中，后面的处理流程从相应的 QUEUE 中取要处理的消息，也把处理结果保存到 QUEUE 中，如此下去，直到完成。

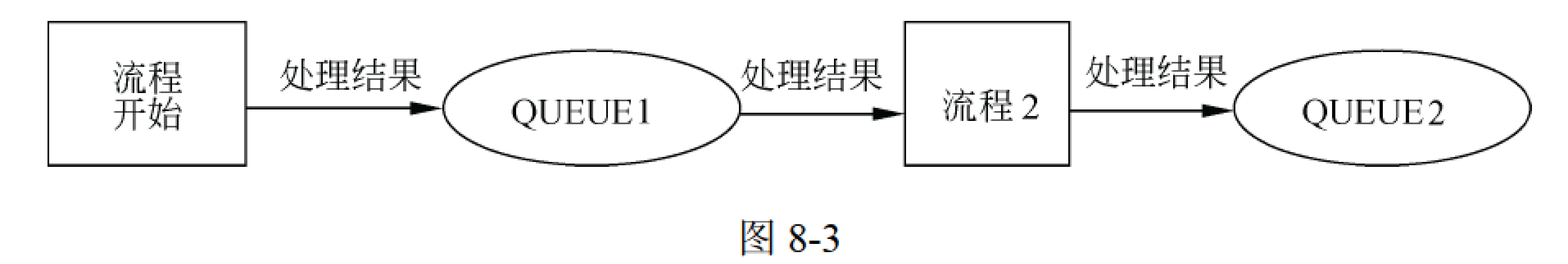


图 8-3

另外，/Q 也常用来做批处理：可以把很多消息发送到一个 QUEUE 中，并设置在某个时间这些消息才生效，再把这些消息取出，进行批处理。

2. /Q 的配置和管理

/Q 的管理工作包括：QMCONFIG 环境变量的设置，用 qmadmin 或图形化管理工具进行 QUEUE SPACE、QUEUE 的创建及管理。因为/Q 也是一种资源管理器，所以要像数据

库那样在 UBBCONFIG 的 GROUP 中进行配置。在 UBBCONFIG 中还要配置 TMQUEUE、TMQFORWARD 这两个 SERVER。下面分别进行说明：

- QMCONFIG 环境变量的设置

QMCONFIG 环境变量指定存储 QUEUE SPACE 的设备（文件）名，以便 qmamdmin 对它进行管理。它可在环境变量中设置，也可在执行 qmadmin 时在命令行中指定。

在 UNIX 下

示例 8-13：

```
QMCONFIG=/usr/tuxedo/qsample/QUE; export QMCONFIG
qmadmin /usr/tuxedo/qsample/QUE
```

在 NT 下

示例 8-14：

```
SET QMCONFIG=d:\qsample\QUE
qmadmin d:\qsample\QUE
```

QMCONFIG 中指定的设备（文件）要先用 crdl 在 Tuxedo 文件系统中创建。

示例 8-15：

```
D:\>qmadmin
>crdl d:\qsample\QUE 0 5000
>q
```

该设备（文件）中可以有多个 QUEUE SPACE。 QUEUE 中的消息等数据就保存在该设备（文件）中。

- qmadmin 的使用方法

Tuxedo 提供一个命令行管理工具 qmadmin，用于对/Q 进行管理，它类似 tmadmin，常用的命令介绍如下。

- **qspacecreate**　创建一个新的 QUEUE SPACE。
- **qcreate**　在某个 QUEUE SPACE 上创建 QUEUE。
- **qinfo**　查看某个 QUEUE 中的信息。
- **qlist**　显示一个 QUEUE SPACE 所包含的 QUEUE，及其中的当前消息个数。
- **qopen**　打开一个 QUEUE SPACE。
- **qclose**　关闭一个 QUEUE SPACE。

说明：在 qmadmin 中输入 help 可以列出所有的命令，用 help 命令名也可以得到其子命令的帮助，举例如下。

示例 8-16：

```
> help qinfo

qinfo [queue name]
-------------------
List information about the specified queue or for all queues.
```

```
This command lists the number of messages on the specified
queue or all queues if no argument is given, and the amount
of free space in the queue space. In verbose mode, this
command also lists the queue creation parameters for each queue.
```

1. QUEUE SPACE 的创建

在 qmadmin 中执行 qspacecreate，按提示进行操作，说明如下。

示例 8-17：

```
> qspacecreate
Queue space name: myqueuespace
IPC Key for queue space: 230458
Size of queue space in disk pages: 200
Number of queues in queue space: 3
Number of concurrent transactions in queue space: 3
Number of concurrent processes in queue space: 3
Number of messages in queue space: 12
Error queue name: errq
Initialize extents (y, n [default=n]):
Blocking factor [default=16]: 16
```

参数说明如下。

（1）--IPC Key for queue space：

该 QUEUE SPACE 的 IPC KEY，范围为 32 768～262 143，并且不要和系统的其他 IPC 资源的 ID 号冲突。

（2）--Size of queue space in disk pages：

该 QUEUE SPACE 的大小，以页为单位。

（3）--Number of queues in queue space：

该 QUEUE SPACE 中最多可以有多少个 QUEUE 在里面。

（4）--Number of concurrent transactions in queue space：

在该 QUEUE SPACE 中最多可以有多少个事务同时存在，计算方法如下。

① 该 GROUP 中的每个 TMS_QM 会用到一个事务。

② 该 GROUP 中的 TMQEUE，TMQFORWARD 也会各自用到一个事务。

③ qmadmin 会用到一个事务。

④ 客户端在调用 tpenqueue()、tpdequeue()之前开始的事务也要计算上，要估计最多可能有多少个这样的客户端同时使用该 QUEUE SPACE。

（5）--Number of concurrent processes in queue space：

最多可以有多少个进程同时存取该 QUEUE SPACE，要包括 TMQUEUE、TMFORWARD 这两个进程。

（6）--Number of Messages in queue space：

该 QUEUE SPACE 中最多可以有多少个 MESSAGE 在里面。

（7）--Error queue name：

当一个消息被回滚达到指定次数，Tuxedo 把该消息发送到 ERROR QUEUE 中，如果输入了 ERROR QUEUE NAME，则必须用 qcreate 创建该 QUEUE，如果没有创建 ERROR QUEUE，那么本来应该发送到 ERROR QUEUE 中的消息将被丢弃。

在创建 ERROR QUEUE 时，以下几个参数不能设置。

示例 8-18：

```
Queue order (priority, time, fifo, lifo):
Out-of-ordering enqueing (top, msgid, [default=none]):
Retries [default=0]:
Retry delay in seconds
```

（8）--Initialize extents (y, n [default=n])：

是否初始化该 QUEUE SPACE 的存储空间。

> **注 意**
>
> 因为在创建 QUEUE SPACE 过程中会用到 IPC 资源，所以如果在创建 QUEUE SPACE 时失败，在重新创建之前，最好把这些 IPC 资源释放掉，可在 qmadmin 中用 ipcrm 命令释放某个 QUEUE SPACE 所占用的 IPC 资源。

2. QUEUE 的创建

示例 8-19：

```
> qcreate
Queue name: service1
Queue order (priority, time, fifo, lifo): fifo
Out-of-ordering enqueuing (top, msgid, [default=none]): none
Retries [default=0]: 2
Retry delay in seconds [default=0]: 30
High limit for queue capacity warning (b for bytes used, B for blocks used,
% for percent used, m for messages [default=100%]): 80%
Reset (low) limit for queue capacity warning [default=0%]: 0%
Queue capacity command:
No default queue capacity command
Queue 'service1' created
```

参数说明如下。

（1）--Queue name：

要创建的 QUEUE 的名称。

（2）--Queue order (priority, time, fifo, lifo)：

发送到该 QUEUE 的消息的存取顺序。

① priority：按优先级（在 tpenqueue()的 TPQCTL 参数中指定）。

② time：按时间（在 tpenqueue()的 TPQCTL 参数中指定）。

③ fifo：先进先出。

④ lifo：先进后出。

（3）--Out-of-ordering enqueuing (top, msgid, [default=none]):

指定一个消息可以放在该 QUEUE 的最前面，或在某个 MSGID 之前。

（4）--Retries [default=0]: 2

默认情况下，当从一个 QUEUE 中取出某个 MESSAGE 的事务回滚时，该 MESSAGE 会被重新放回到该 QUEUE 中，当该消息又处于该 QUEUE 的最前面时，TMQUEUE 将再次试图取出该消息，这里指定重试的次数，默认值为零，也就是不进行重试。当达到重试的次数时，如果该 QUEUE SPACE 设置了 ERROR QUEUE，那么这个消息将被移到该 ERROR QUEUE 中，如果该 QUEUE SPACE 没有设置 ERROR QUEUE，那么这个消息将被丢弃。

（5）--Retry delay in seconds [default=0]: 30

指定重试的时间间隔。

（6）--High limit for queue capacity warning (b for bytes used, B for blocks used, % for percent used, m for messages [default=100%]): 80%

（7）--Reset (low) limit for queue capacity warning [default=0%]: 10%

（8）--Queue capacity command: /usr/app/bin/mailme myqueuespace service1

以上 3 个设置，当该 QUEUE 中的消息对空间的使用或消息数达到设定的阀值时，Tuxedo 系统自动执行一个命令，以达到对该 QUEUE 中的消息进行自动处理的目的。

在以上的设置中，当该 QUEUE 中 80%的空间被使用时，将执行/usr/app/bin/mailme, myqueuespace service1 是 mailme 的参数。

当第一次到达 80%之后，/usr/app/bin/mailme 被执行，只有当降为 10%之后，又到达 80%时，/usr/app/bin/mailme 才再次被执行。

（9）--Reply Queue 和 Failure Queue

当采用转发方式时，TMQFORWARD 把用 tpcall()调用的与该 QUEUE 同名的 SERVICE 的处理结果放到 REPLY QUEUE 中，该 REPLY QUEUE 的名字在 tpenqueue()中指定，如果该 SERVICE 处理失败，TPRETURN(TPFAIL….)，那么 TMQFORWARD 将把错误信息写到 FAILUER QUEUE 中，如果没有创建 Reply QUEUE 或 Failure Queue，TMQFORWARD 将把该 SERVICE 的返回丢弃，调用 tpdequeue()的客户端将收不到任何信息，如果创建了 REPLY QUEUE，那么即使该 SERVICE 没有返回信息，TMQFORWARD 也会往该 REPLY QUEUE 中写入一条长度为零的消息，客户端可以收到该消息。

3. UBBCONFIG 中要做的配置

（1）GROUP 中的配置

在 GROUP 中的配置与数据库通过 XA 协议与 Tuxedo 连接的配置差不多， 因为 QUEUE SPACE 是资源管理器，而一个组只能有一个资源管理器。所以 QUEUE SPACE 与 QUEUE SERVER GROUP 之间是一对一的关系， 在 GROUP 中的配置如下。

① TMS(TRANSACTION MANAGEMENT SERVER): TMS_QM。

② OPENINFO，它的设置格式如下。

示例 8-20：

```
OPENINFO="Tuxedo/QM:<device_name>:<queue_space_name>"
```

Tuxedo/QM：为/Q 所对应资源管理器的名称，在$TUXDIR/udataobj/RM 指定。

device_name：指定存储该 QUEUE SPACE 的设备（文件）名。

queue_space_name：为该 QUEUE SPACE 的名称。

在 UNIX 下示例如下。

示例 8-21：

```
*GROUPS
QUE1
LMID = SITE1 GRPNO=2
TMSNAME = TMS QM TMSCOUNT = 2
OPENINFO = “Tuxedo/QM:/home/QUE:QSPACE”
```

在 NT 下示例如下。

示例 8-22：

```
*GROUPS
QUE1
LMID = SITE1 GRPNO=2
TMSNAME = TMS_QM TMSCOUNT = 2
OPENINFO = “Tuxedo/QM:d:\qsample\QUE;QSPACE”
```

（2）在 SERVER 中的配置

在 SERVER 这一节中要配置 TMQUEUE（必须）、TMQFORWARD（可选）这两个 SERVER。

TMQUEUE 的设置格式如下如下。

示例 8-23：

```
TMQUEUE CLOPT="-s QSPACENAME:TMQUEUE --[-t  timeout]"
```

“-s”：指定该 SERVER 要发布的 SERVICE 的名称，采用别名方式，用 QUEUE SPACE 的名字加上 TMQUEUE；

“-t”：非事务中的/Q 操作的超时时间。

可参考下面的例子。

示例 8-24：

```
*SERVERS
TMQUEUE SRVGRP = QUE1 SRVID = 1
CLOPT = "-s QSPACE:TMQUEUE -- -t 60"
```

TMQFORWARD 的设置格式如下。

示例 8-25：

```
TMQFORWARD CLOPT="-- -q qname[,qname...] [-t trantime]
[-i idletime] [-e] [-d] [-n][-f delay] "
```

“-q”：“-q qname[，qname...] ” 是用 “，” 隔开的 QUEUE 的名称，TMQFORWARD 将从这些 QUEUE 中取数据，并用 tpcall()调用与该 QUEUE 同名的 SERVICE 进行后续处理。

“-t”：TMQFORWARD 调用 tpbegin()时指定的事务超时时间，默认值为 60 秒。

“-i”：指定当该 QUEUE 中的消息都已被取出后，隔多长时间，TMQFORWARD 再次读该 QUEUE 看是否有新的消息到来。

“-e”：如果在这些 QUEUE 中都没有消息，那么 TMQFORWARD 将退出。

“-d”：删除导致所调用的服务失败，但返回了响应的原始消息。使它不用再被重试。

“-n”：当 TMQFORWARD 调用 tpcall()时，所用的 FLAG 为 TPNOTRAN。

“-f”：“-f delay” 指定 TMQFORWARD 发送异步服务请求的时间间隔。

可参考下面的例子。

示例 8-26：

```
*SERVERS
TMQFORWARD SRVGRP=QUE1 SRVID = 5
CLOPT="-- -i 2 -q STRING"
```

8.3　动态配置 tmconfig

8.3.1　概述

tmconfig 是一个交互式工具，可以用来动态修改 Tuxedo 运行环境配置。

8.3.2　配置 tmconfig 运行环境

示例 8-27：

```
TUXDIR=/home/landingbj/software/tuxedo11g
TUXCONFIG=/home/landingbj/test/tuxconfig
EDITOR=vi
```

当输入 tmconfig 出现以下输出时表示环境设置正确。

示例 8-28：

```
$ tmconfig
   Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
    5)SERVICES  6) NETWORK 7) ROUTING q) QUIT 9) WSL
    10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]:
```

8.3.3 tmconfig 常用操作

1. 用 tmconfig 增加新主机

（1）进入 tmconfig 后，选择 2）MACHINES 项。
（2）然后可以先选择 3）RETRIEVE 查看当前的配置，缺省是第一个 MACHINE 的配置。
（3）通过选择 2）NEXT 可以一直向后搜索 MACHINE 配置，直到空记录为止。
（4）选择 4）ADD。
示例 8-29：

```
Enter editor to add/modify fields [n]? y
```

（5）进入 vi 编辑状态，可以按照一定格式增加配置。
格式为：MIB 域名[tab]值
（6）增加 MACHINE 必须加入以下的域。

- oTA_TUXCONFIG
- oTA_TUXDIR
- oTA_APPDIR
- oTA_TLOGDEVICE
- oTA_TLOGSIZE
- oTA_PMID
- oTA_LMID
- oTA_TYPE

示例 8-30：

```
TA TUXCONFIG /home/landingbj/test/tuxconfig
TA TUXDIR /home/landingbj/software/tuxedo11g
TA APPDIR /home/landingbj/test
TA TLOGDEVICE /home/landingbj/test/TLOG
TA ULOGPFX /home/landingbj/test/ULOG
TA_ENVFILE /home/landingbj/test/ENVFILE
TA_TLOGSIZE 150
TA PMID SERVER109
TA LMID SITE1
TA_TYPE Sun
```

（7）存盘退出 vi，执行操作即可。
（8）激活新增的 MACHINE：重新选择 2）MACHINES –> 5)UPDATE。
（9）进入 vi 后查找 TA_STATE，将其值从 NEW 改为 ACTIVE。
（10）存盘退出 vi 并执行操作。

2. 用 tmconfig 增加新 Server 进程

（1）进入 tmconfig 后，选择 2）SERVER 项。

（2）可以先选择 3）RETRIEVE 查看当前的配置，缺省是第一个 SERVER 的配置。

（3）通过选择 2）NEXT 可以一直向后搜索 SERVER 配置，直到空记录为止。

（4）选择 4）ADD。

示例 8-31：

```
Enter editor to add/modify fields [n]? y
```

（5）进入 vi 编辑状态，可以按照一定格式增加配置。

格式为：MIB 域名[tab]值

（6）增加 MACHINE 必须加入以下的域。

- oTA_SERVERNAME
- oTA_SRVGRP
- oTA_SRVID

示例 8-32：

```
TA SERVERNAME/home/landingbj/test /teller server
TA_SRVGRP GROUP1
TA_SRVID 15
```

（7）存盘退出 vi，执行操作即可。

8.4　TSAM

8.4.1　TSAM 简介

Oracle TSAM（Tuxedo System Application Monitor）最早可以使用在 Tuxedo10g 版本上，可以对 Tuxedo 系统提供全面的检测以及产生 Tuxedo 系统和应用的详细报告。

本节将介绍如何在 Tuxedo11g 上安装部署 TSAM11g 以及简单的对 Tuxedo 的监控方法。

8.4.2　TSAM 安装

TSAM 可以安装部署在 Windows 或 UNIX 系统上，并可支持 3 种安装方式。

（1）图形化界面（GUI）模式安装。

（2）控制台模式安装。

（3）静默安装。

安装过程中需要选择监控数据存放的数据库（Oracle/Derby）、管理控制台使用的应用服务器（WebLogic/Tomcat），并设置管理员密码。

以下以图形化界面（GUI）模式在 Linux 上安装 TSAM 为例进行演示，控制台模式与 GUI 模式相类似。

（1）登录 Linux 系统，进入 TSAM 安装程序所在目录，执行下面语句。

示例 8-33：

```
"$ ./tsam11gR1_64_Linux_01_x86.bin"
```

系统将出现如图 8-4 所示界面。

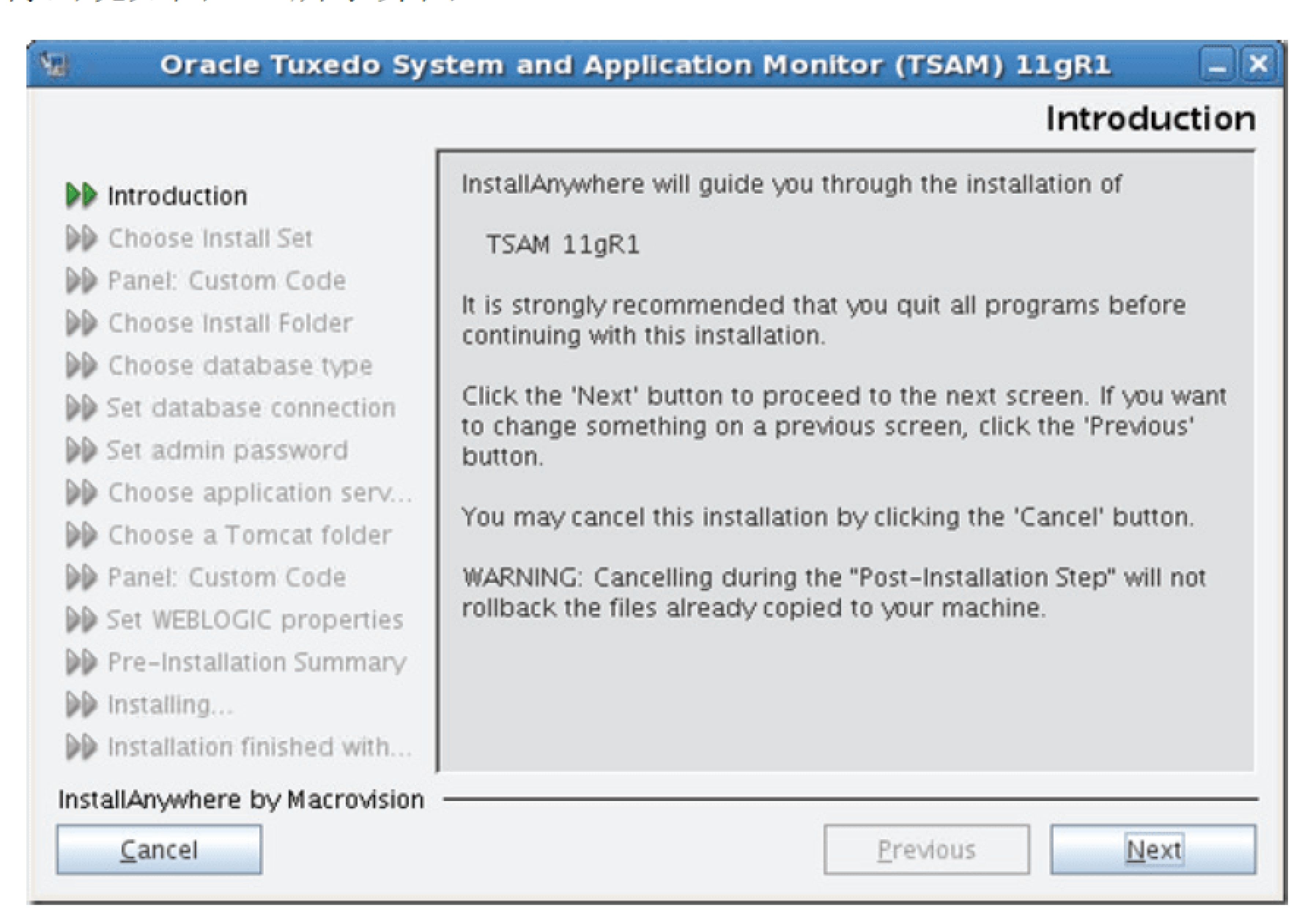


图 8-4

（2）单击 NEXT 按钮，出现如图 8-5 所示界面。

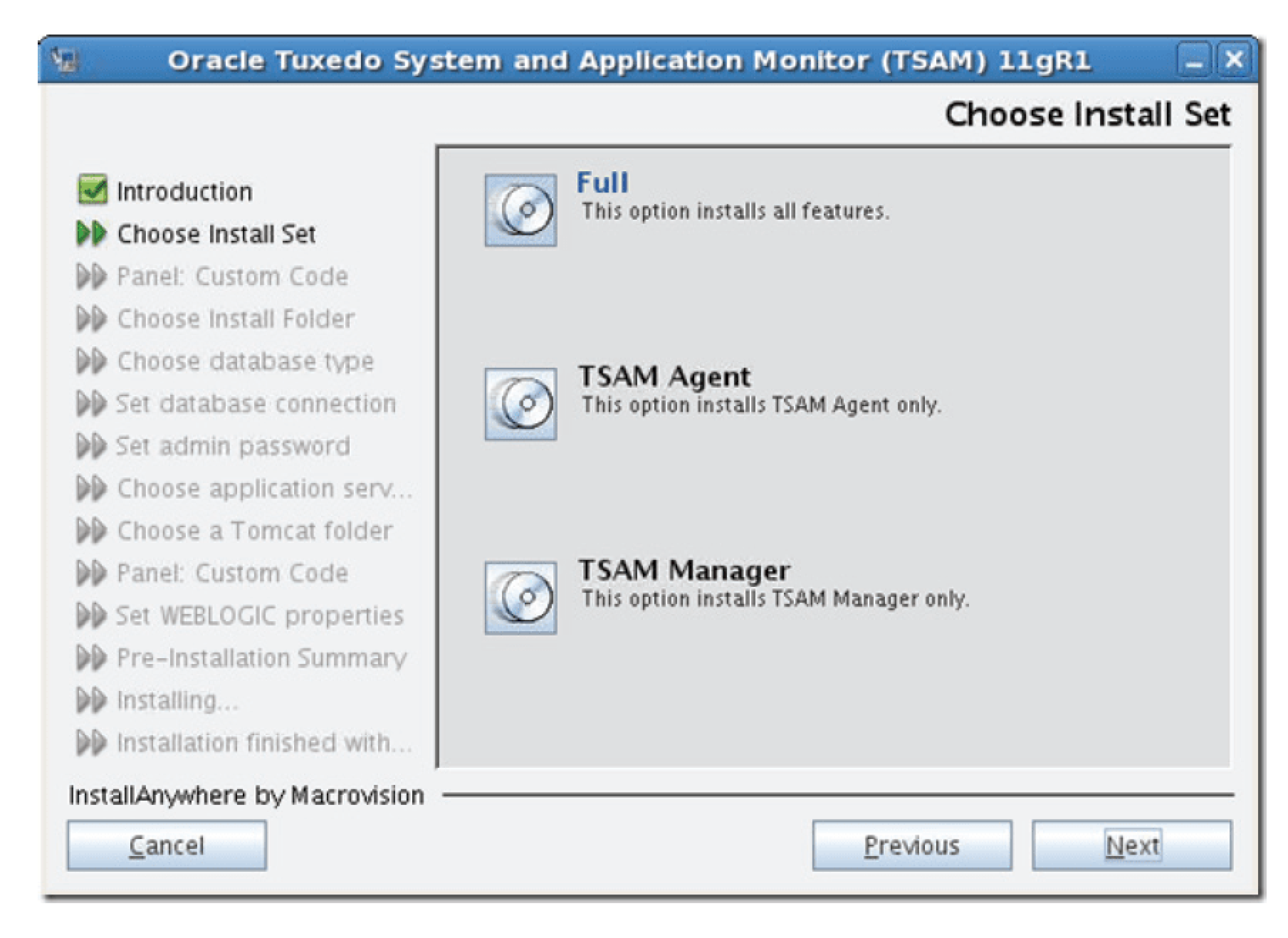


图 8-5

（3）选择“FULL”选项，并单击 Next 按钮，出现如图 8-6 所示界面。

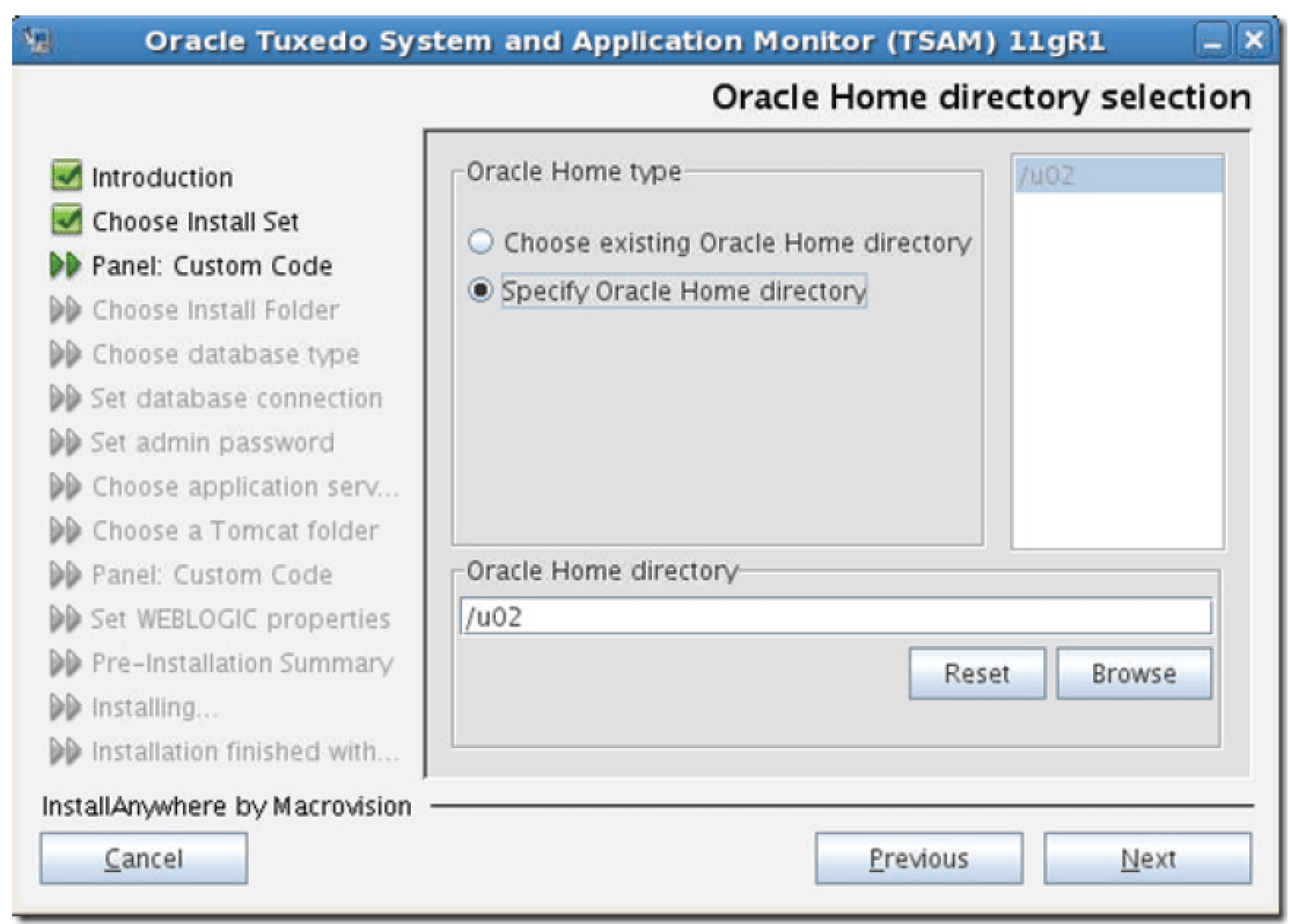


图 8-6

（4）初次安装需要指定 Oracle 程序安装路径，例如，输入“/U02”，单击 Next 按钮，出现如图 8-7 所示界面。

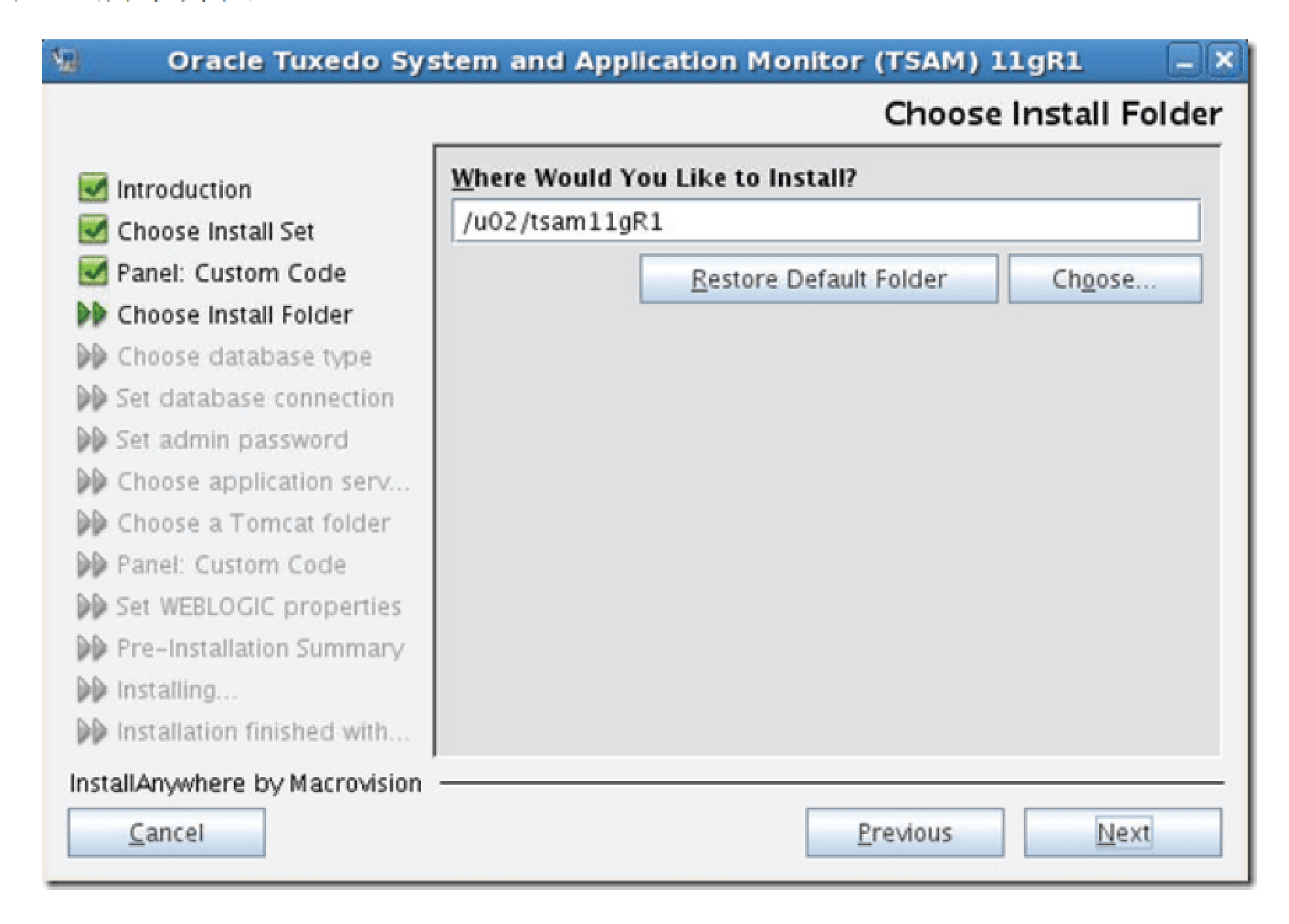


图 8-7

（5）需要再次指定 TSAM 安装路径，例如，输入“/u02/tsam11”，单击 Next 按钮，出现如图 8-8 所示界面。

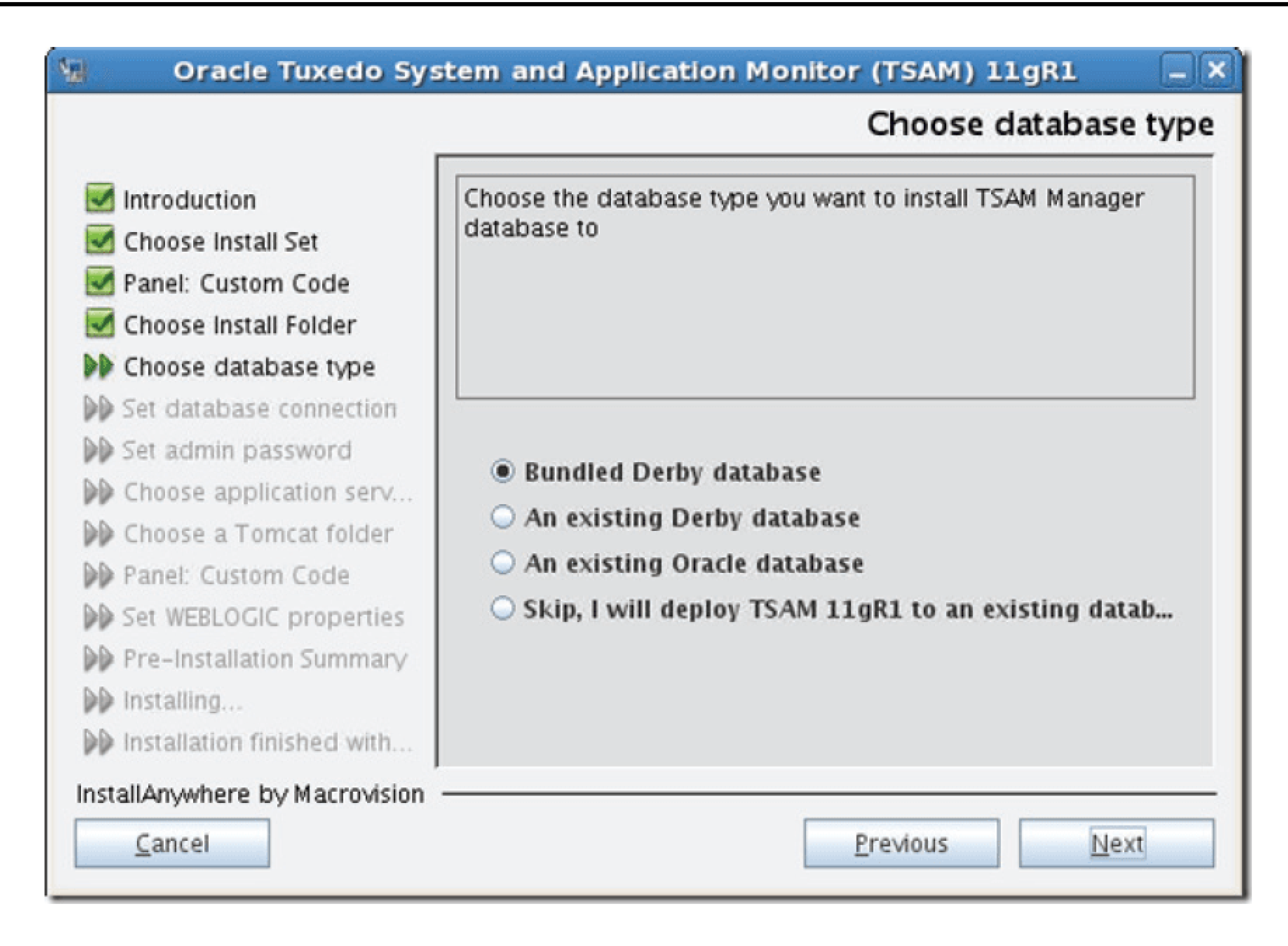


图 8-8

（6）因为 TSAM 的数据需要存储在数据库中，这里选择 Bundled Derby database 单选按钮，并单击 Next 按钮，出现如图 8-9 所示界面。

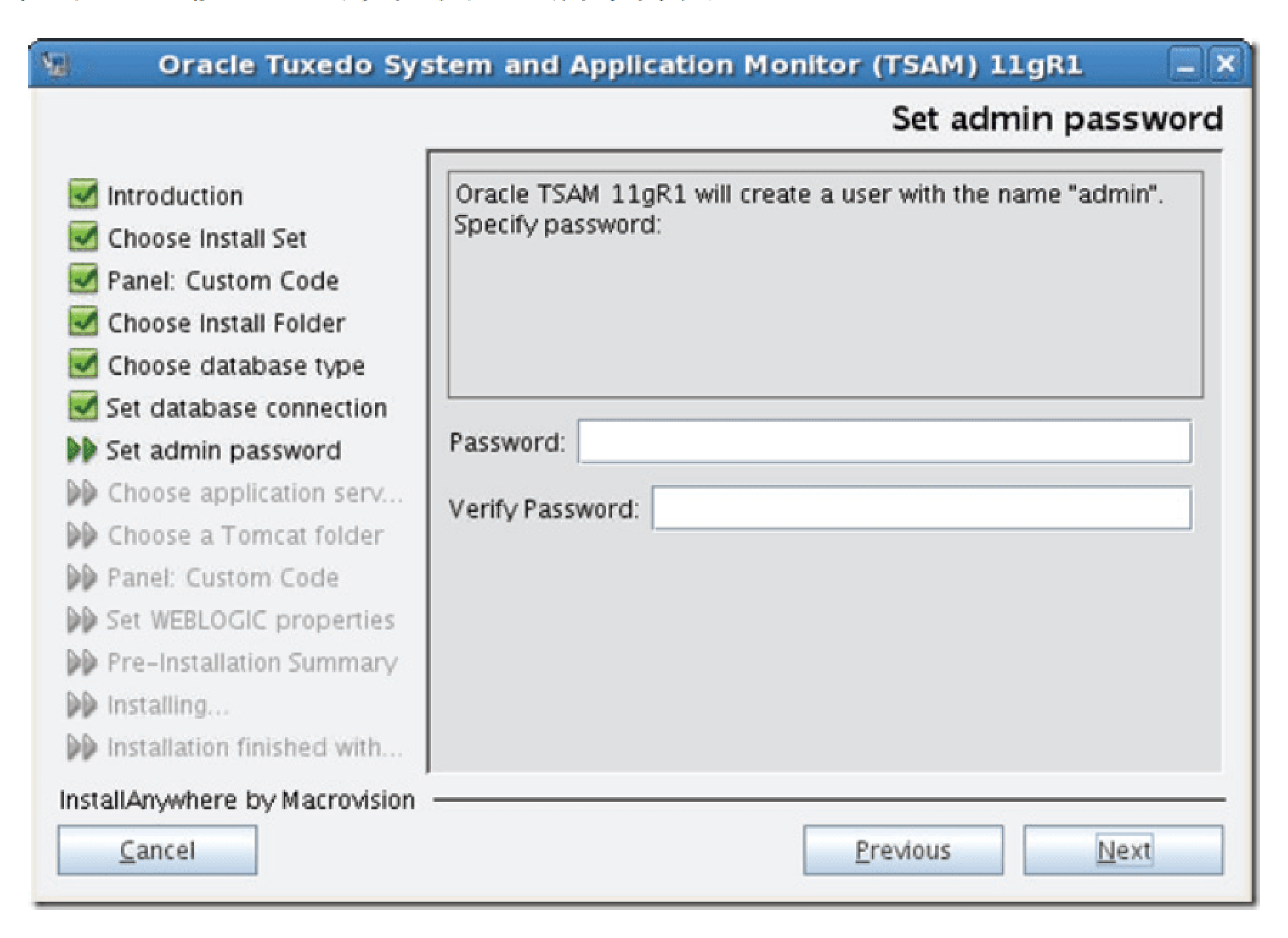


图 8-9

（7）再次输入 TSAM 的验证密码，在控制台登录时需要用到此密码，输入完成后单击 Next 按钮，出现如图 8-10 所示界面。

（8）选择应用服务器，按照默认安装 tomcat 服务器，单击 Install 按钮，出现如图 8-11 所示界面。

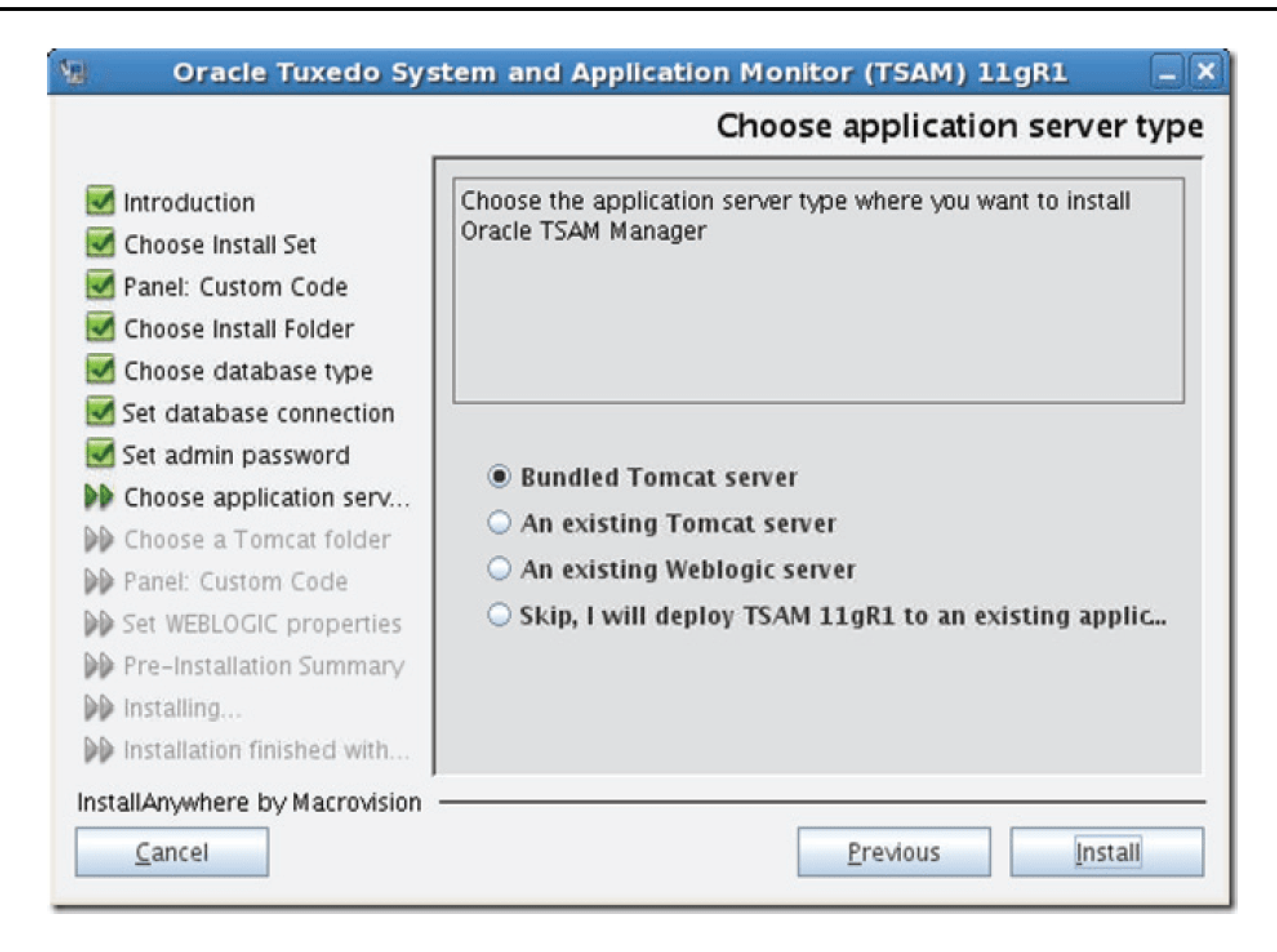


图 8-10

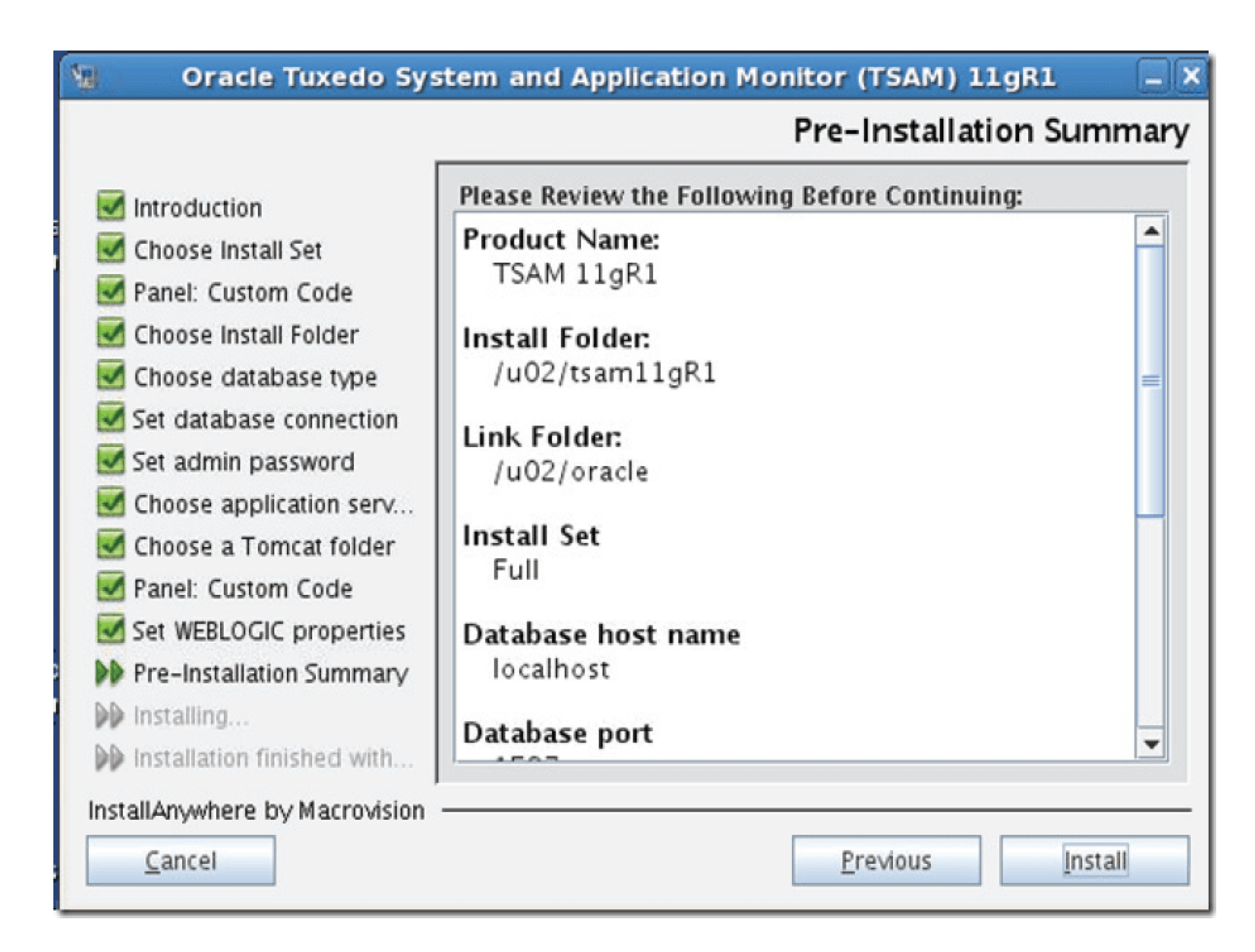


图 8-11

（9）这里是总结，再次单击 Install 按钮就开始安装，并出现如图 8-12 所示界面。

（10）安装完成之后，出现安装总结，如果出现错误就针对错误进行检查，没有错误就单击 Next 按钮结束安装，如图 8-13 所示。

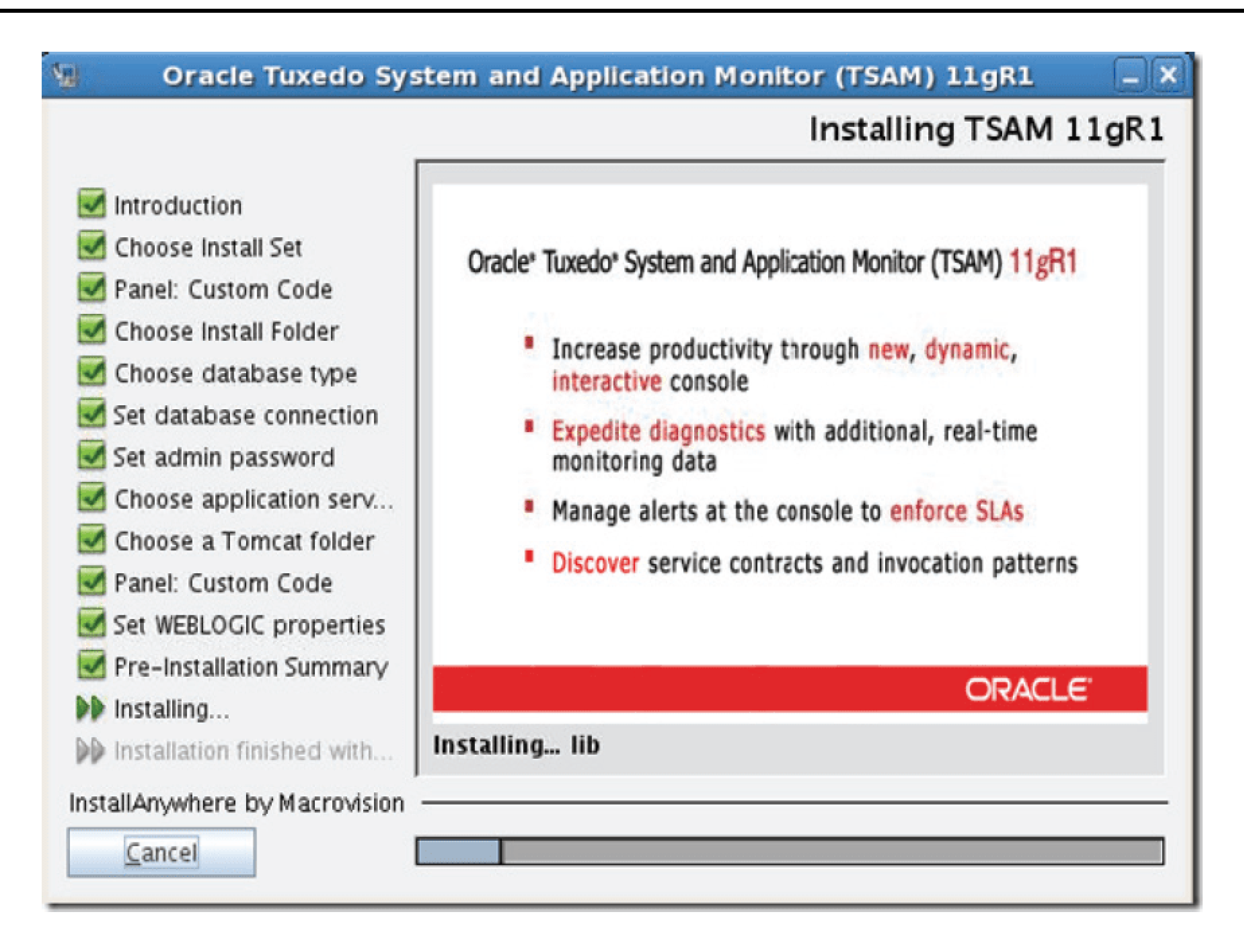


图 8-12

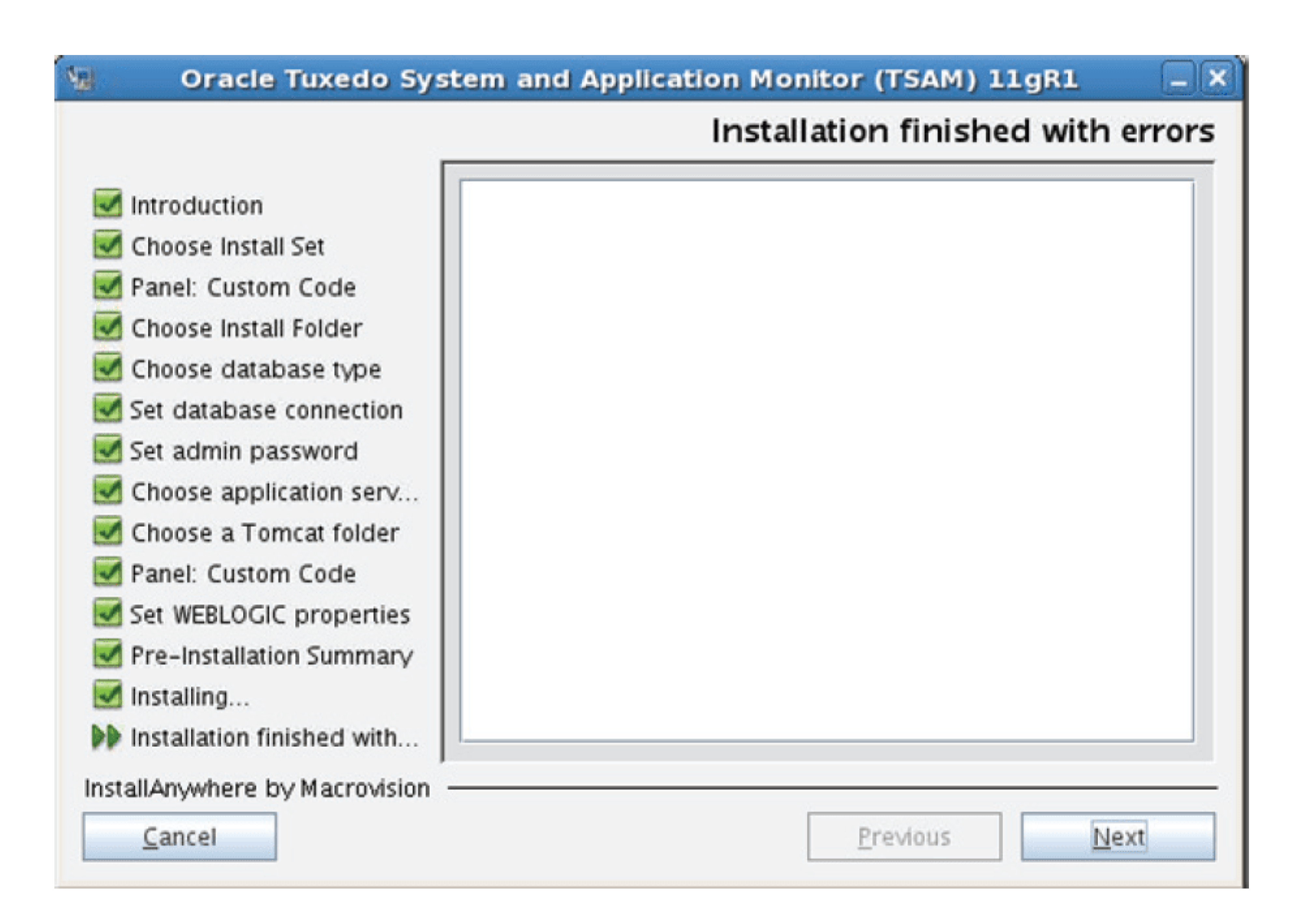


图 8-13

8.4.3 TSAM 配置

（1）配置 LMS 服务器：为了使 TSAM 能够监控 Tuxedo 系统，需要在 Tuxedo 中配置 LMS 服务器，如下所示。

示例 8-34：

```
LMS  SRVGRP=GROUP1  SRVID=20  MINDISPATCHTHREADS=1  MAXDISPATCHTHREADS=5
CLOPT="-A -- -l 10.10.124.237:8080/tsam"
```

（2）启动 TSAM：进入 TSAM 安装目录下的 bin 目录，执行脚本“./startup.sh”即可启动 TSAM。

（3）登录 TSAM 控制台：打开浏览器，在地址栏中输入“http://localhost:8080/tsam”回车就可以登录 TSAM 的控制台。输入用户名 admin，密码为安装时设置的管理员密码。单击 login 按钮登录控制台，如图 8-14 所示。

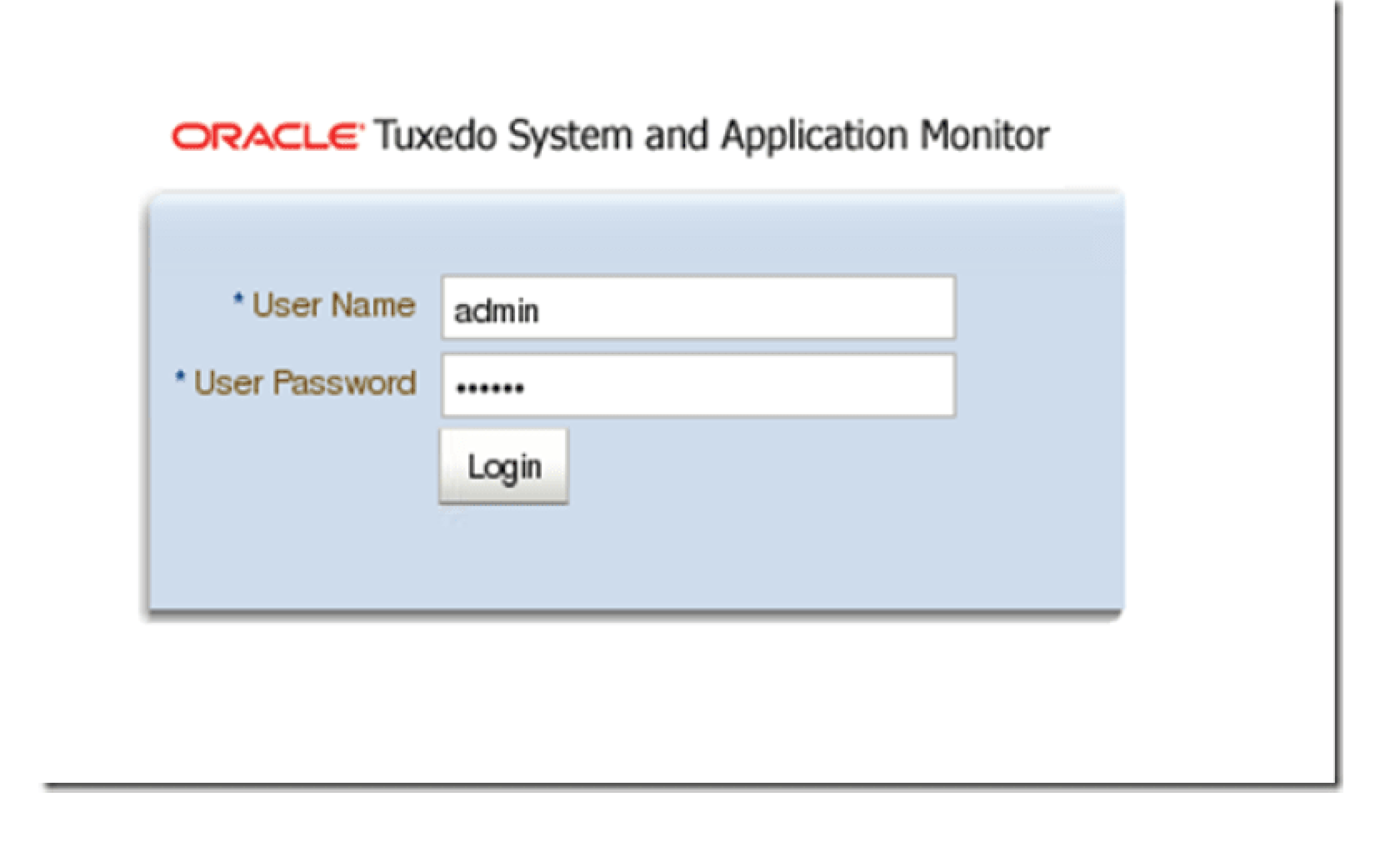


图 8-14

8.4.4　TSAM 监控

（1）在对 Tuxedo 进行监控之前，需要先配置监控策略，选择 policy 下拉菜单中的 Tuxedo monitoring policy 命令，打开 Monitoring Policy List 对话框，如图 8-15 所示。

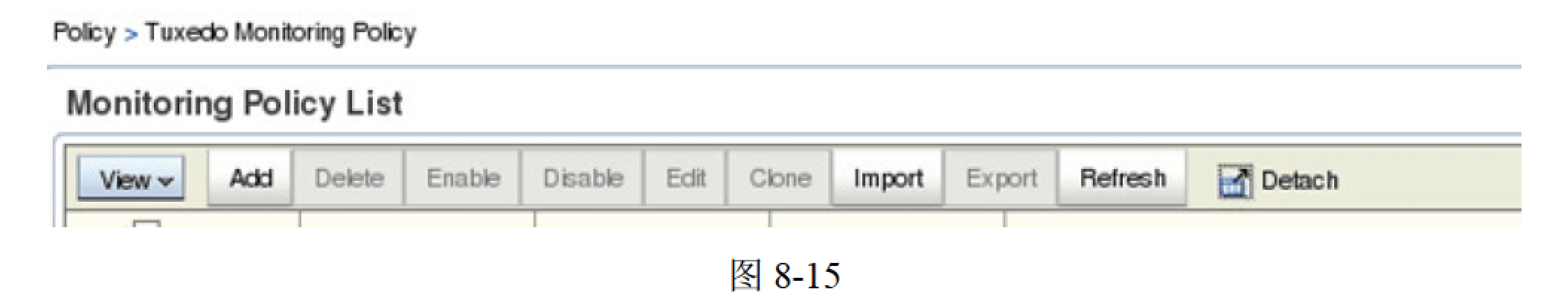


图 8-15

（2）单击 add 按钮，按要求输入策略名，打开左侧 Tuxedo component 栏下拉菜单，选中所需要监控的 Tuxedo 应用 SERVER，在右侧选中 SERVICE 选项卡，选中 Enable 表示启用服务监控，选中 ratio 单选按钮，然后在 SERVICE 下拉列表框中选中所需要监控的服务，单击 Save&Enable 按钮保存并启用，如图 8-16 所示。

（3）配置好监控策略后，就可以对所选 Tuxedo SERVICE 进行监控，选择 Tuxedo metrics 下拉菜单中的 SERVICE 命令，就会出现 SERVICE 监控页面，在左侧 SERVICE

selection 选项区域中依次选中监控策略中定义的 SERVER，在 monitoring mode 下拉列表框中选择 Most Action（live）选项，可以实时对 SERVICE 进行监控，右侧下拉菜单表示监控选项，包括 SERVICE 执行时间、调用次数等，这里选择 Execution Time 选项。然后在客户端进行调用之后，这里就会显示出 SERVICE 的处理情况，如图 8-17 所示。

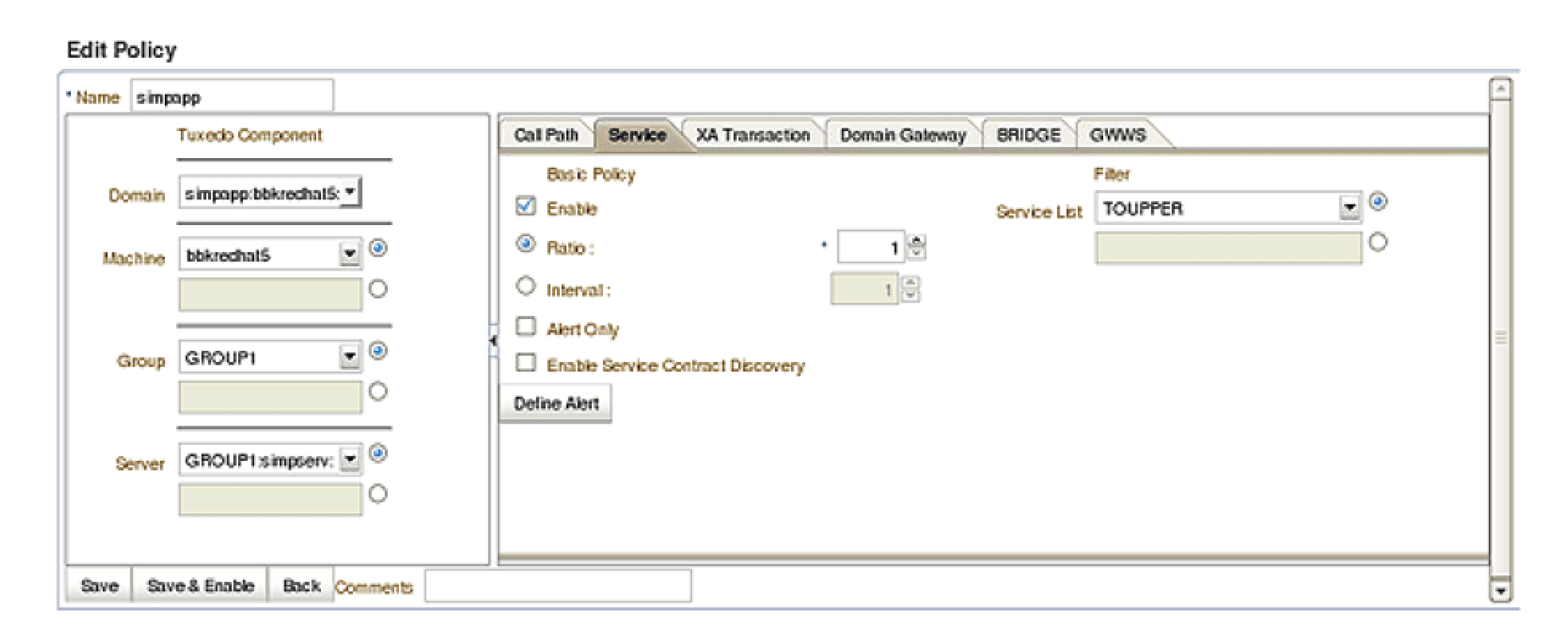


图 8-16

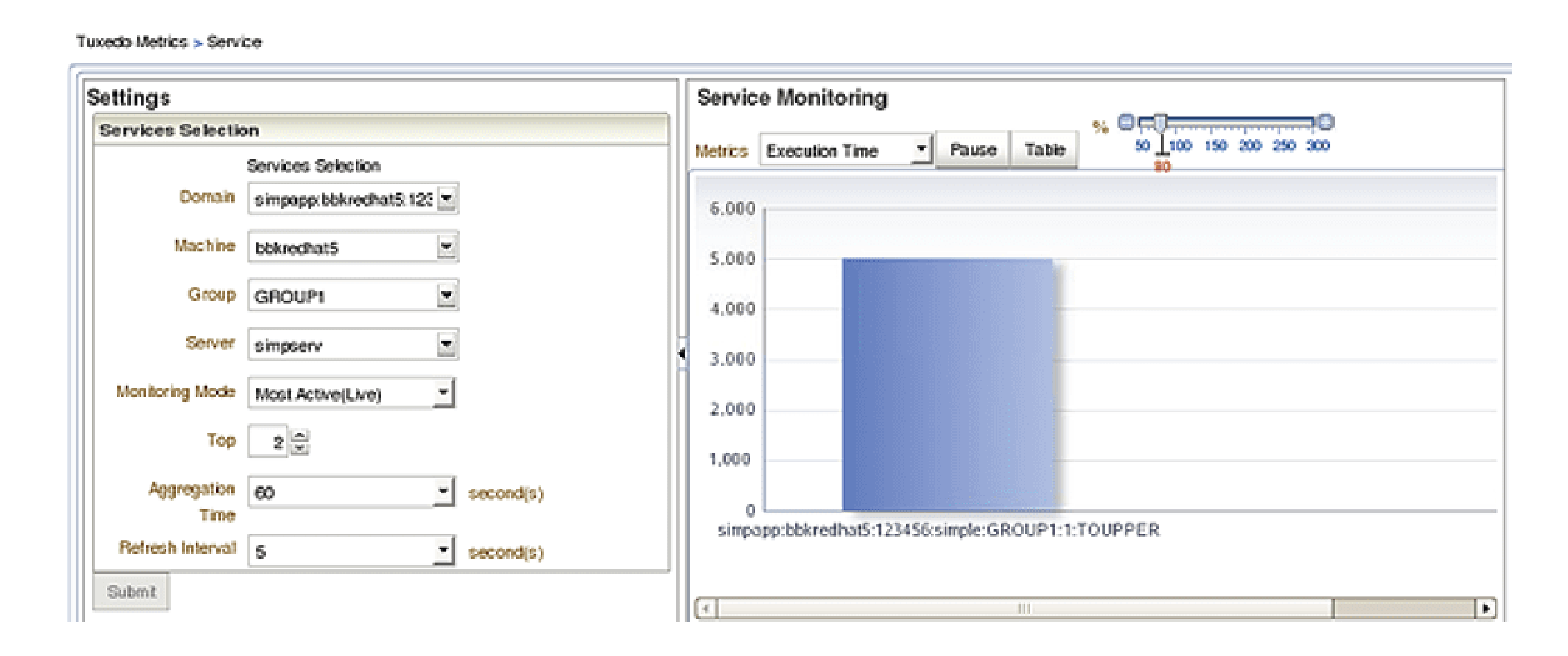


图 8-17

8.4.5 TSAM 监测预警

在 TSAM 中还提供了监测预警的功能，可以通过以下方式实现。

（1）单击 alert 下拉菜单，选择 Tuxedo Alert Definition 命令，打开预警定义列表，如图 8-18 所示。

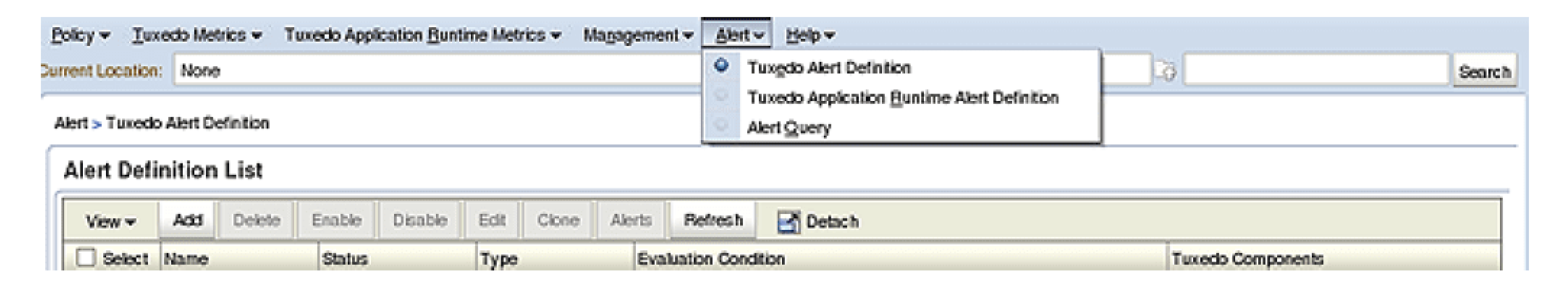


图 8-18

（2）单击 Add 按钮，出现 Define New Alert 对话框，如图 8-19 所示，按要求部署，就可以新增一个预警。

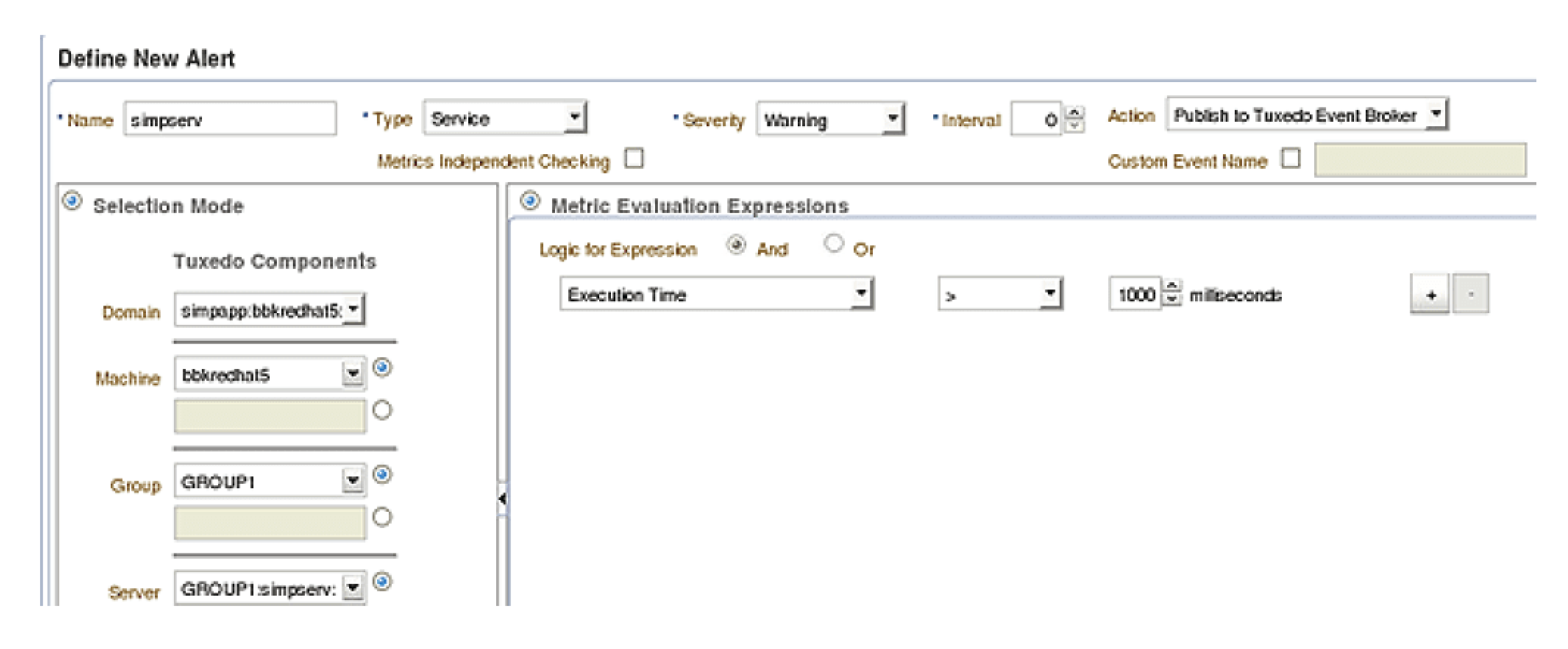


图 8-19

（3）在上面一栏中填入 name 为预警的名字，type 为类型，这里选择 SERVICE，严重性选择警告，间隔按默认，动作选择发布到事件代理。

（4）下面一栏左侧 Tuxedo components 按顺序选中需要预警的 SERVER。右侧预警指标表达式中选择为 Execution Time>1000 毫秒，即执行时间超过 1 秒就会发出告警，单击保存并启用。

（5）程序执行后，单击 Alert 下拉菜单，选择 Alert Query 命令，出现警告查询的页面，在这里就可以看出预警监测发出的警告信息，如图 8-20 所示。

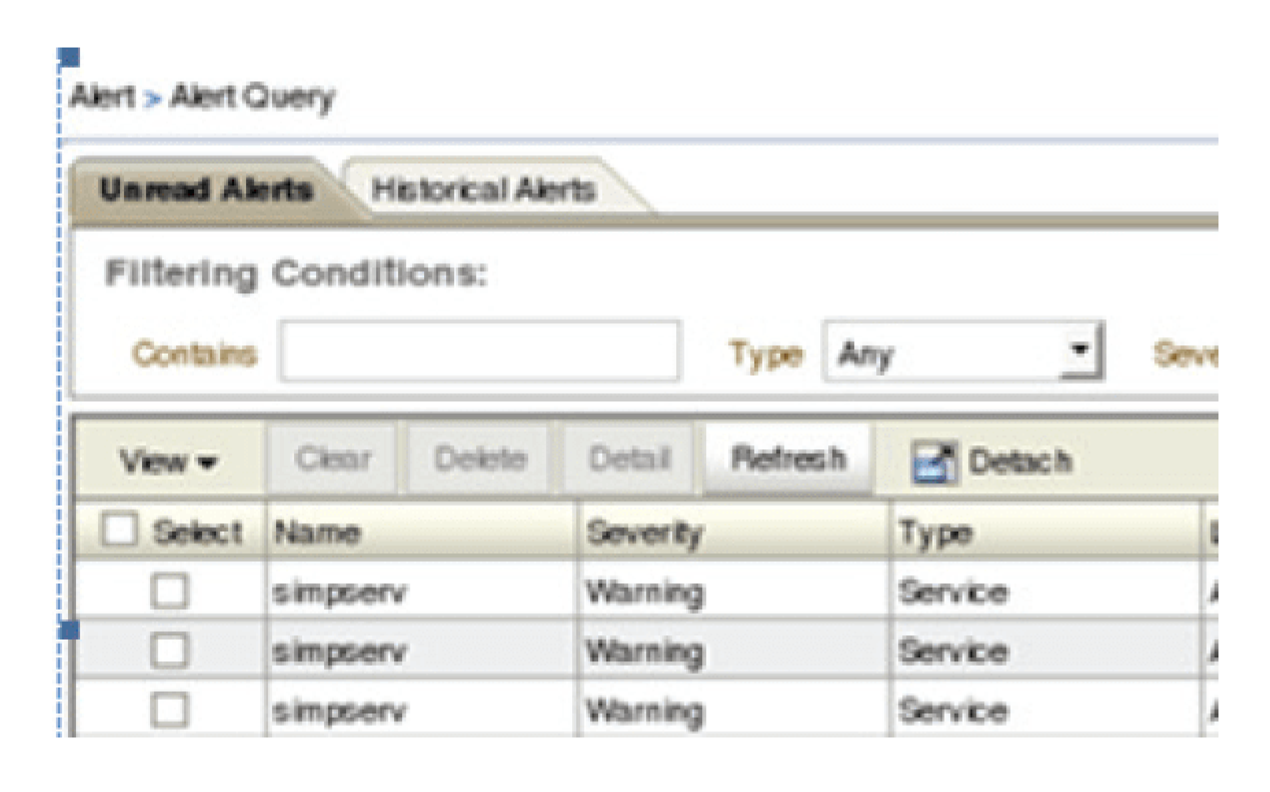


图 8-20

8.5　高可用性

8.5.1　高可用性概述

Tuxedo 作为电子商务交易平台，允许客户机和服务器参与一个涉及多个数据库协调更新的事务，并能够确保数据的完整性。Tuxedo 能够保证对电子商务系统的不间断访

问。它可以对系统组件进行持续的监视，查看是否有应用系统、交易、网络及硬件的故障。一旦出现故障，Tuxedo 会从逻辑上把故障组件排除，然后进行必要的恢复性步骤。

Tuxedo 可以根据系统的负载指示，自动开启和关闭应用服务，并可以均衡所有可用系统的负载，以满足对应用系统的高强度使用需求。借助 DDR（数据依赖路由），Tuxedo 可按照消息的上下文来选择消息路由。其队列功能，可使分布式应用系统以异步“少连接”方式协同工作。

Tuxedo LLE 安全机制可确保用户数据的保密性，应用/交易管理接口（ATMI）为 50 多种硬件平台和操作系统提供了一致的应用编程接口。

Tuxedo 基于网络的图形界面管理可以简化对电子商务的管理，为建立和部署电子商务应用系统提供了端到端的电子商务交易平台。

8.5.2 高可用性详细分析

以下子模块是 Tuxedo 为实现高可用性所提供的功能模块。

1．管理模块

Tuxedo 提供了实现分布式管理的架构。该架构运用一个 manage/agent 模型，以及运用管理信息库（Management Information Base，MIB）来实现 Tuxedo 的分布式管理。

MIB 能够让应用管理员动态集中管理应用所涉及的硬件、软件以及网络资源。应用设计人员可以指定 SERVER 和 SERVICE 在哪启动，同样可以指定当进程出现错误时迁移的地方；可以设定属性值，包括调度信息、服务恢复准则、超时时间等。

在 MIB 上实现的管理接口包括综合的命令行工具、GUI 工具、SNMP 代理。

另外，MIB 允许所有系统参数的查询（以及修改，如果可以修改），并且提供了所有的系统事件，当 Tuxedo 运行环境有重大改变时这些事件给出通知。

Oracle Tuxedo 有一套内部机制来支持运行时应用高可用性。

（1）BBL--Bulletin Board Liaison 是一个节点监控，负责在一个节点上监控所有进程（应用的以及管理的）。

（2）DBBL--Distinguished BBL 是一个主节点监控，负责监控集群中的每个节点，应当为一个网络应用建立一个备机 DBBL。

（3）BRIDGE—提供了集群内部节点的交互。

（4）TMS--Transaction Management Server 专注于事务相关的 DBMS、MQ queue 等的协调。

2．故障检查以及自动恢复

（1）运行时工具

Tuxedo 运行时管理提供自动检测和修改软件错误，完成这些功能的主要工具如上所述。节点服务器、网络连接、应用服务、客户端以及 Tuxedo 管理服务器（包括 BBL/DBBL/BRIDGE）都能被监控，另外，能自动执行尝试改正错误而不要操作人员介入。应用报错能够被日志系统抓获，从而扩展了错误检测范围以及使应用级别恢复成为可能。所有错误

都记录在 Tuxedo 错误日志中(ULOG)，日志文件每个节点都维护一个，但可以通过 Manager Log Central 集中监控。

（2）节点状态检查

DBBL 维持着一个心跳机制，它能检查每个节点，当某个节点心跳不存在了，主节点会尝试重启该被管节点的 BBL，如果该被管节点不能被重启，那么会被标记为隔离状态(如 unavailable)。Bridge 进程也会引起一个节点被标记为隔离状态，如果出现连接超时以及连接无法重建，由于错误持续时间未知，所以隔离的节点不会自动移除。隔离的出现会记录到系统事件中以及打印到错误日志中。如果该问题是短暂的网络中断，当连接重新建立时，连通性将很快恢复；但如果这个问题很严重，管理员可以通过控制台关闭该节点，其他节点继续服务。

（3）Server 状态检查

BBL 周期性地检查应用服务进程的可用性，如果检测到出现错误，Tuxedo 将中断未完成的事务以及重启该服务进程（如果设置了可以重启）。

（4）管理进程自查

DBBL 和 BBL 自动周期性地互相检查，如果必要，会互相重启。Master 节点宕掉如果需要迁移，要在配置文件中配置。迁移可以是人工的也可以是自动的，自动迁移需要 Oracle Enterprise Manager Grid Control 或者第三方集群工具。

（5）客户端状态

BBL 会检测客户端进程的状态，当出现不正常的中断时会做相应的清理工作。在本地客户端的情况下，由进程的状态决定。本地客户端错误，会由 BBL 维持的回复队列超时检测到；如果是远程客户端连接，通过不活动超时来检测不正常中止，当检测到异常、进程中的连接中断、客户端的信息清理时，用户需重新登录以及重新请求。

（6）网络连接

BRIDGE 进程维护集群多个节点内部的通信，通过综合配置文件配置的多重网络地址实现。BRIDGE 运用高优先级的连接，当出现异常中断时，自动转到稍低优先级的连接，如果高优先级的重新能用了，连接又自动转到高优先级的连接（另外，如果第一个网络地址出现阻塞，BRIDGE 进程会利用在同一个优先级下的第二个网络地址，以在高负载的网络情况下增加吞吐量）。

节点之间的交互可以通过超时设定监控，通过 Bridge 进程维护。一旦发现连接错误，Bridge 进程会尝试重新恢复通信，如果所有的网络都不能重建连接，剩下的节点会继续运行该应用。

（7）事务

如果分布式应用进程启用，会定时地自动监控全局事务，如果超时，事务将取消。事务一旦开启，Tuxedo 将跟踪所有的 DBMS(以及其他 XA 相关的资源，如 JMS、MQ queue)，如果一个异常出现在事务完成提交前的任何时间段或者事务被应用直接取消，事务期间所做的数据库修改将全部取消。

（8）应用错误返回

应用的异常能被日志记载，系统事件将始终如一地报告运行情况，这些既可以是手动的也可以通过系统管理软件自动完成。

3. 异常时维护数据的完整性

Tuxedo 实现了 X/Open 分布式事务（X/Open Distributed Transaction Processing，DTP）规范。使用两阶段提交协议的事务管理器（Transaction Management Servers，TMS）确保了全局事务的原子性。全局事务可以覆盖多个、异构的运行在多个异构节点的数据库。TMS 负责协调事务的提交、回滚和恢复。

4. 异常后的维护工作

Tuxedo 提供了一套工具来维持出现异常后的运作。

5. 恢复操作

恢复性的操作通常在某些情况下自动完成，但更常见的还是通过管理命令来维护。

（1）自动恢复

关于自动恢复，以上很多都是讨论这方面的：从应用、管理进程、客户端进程、网络、事务超时等异常中都有自动恢复。

（2）重启节点

启动或者激活一个节点来恢复异常终止的服务，当一个节点是在线运行的，任何未完成的事务都将自动提交或者回滚。

（3）从可迁移的集群中恢复

通过迁移服务到原始节点（如果原始节点自动重启）来完成可迁移的节点之间的恢复。

（4）客户端进程

客户端进程可能需要登录到恢复节点。

8.6 Tuxedo 如何打补丁

8.6.1 备份

（1）备份应用。

（2）备份配置文件。

（3）tmunloadcf >**。

（4）备份环境文件。

（5）备份 udataobj 文件夹，这里包含有 license。

8.6.2 补丁升级

（1）介质准备，确保版本的一致性。

（2）停止 Tuxedo 应用，包括 tlisten 进程。

（3）确保环境变量中有 TUXDIR。

（4）执行补丁安装命令 install.sh(UNIX)或者 install.exe(Windows)。

（5）安装完成检查补丁级别，tmadmin-version。

（6）如补丁回退，执行 uninstall.sh(UNIX)或者 uninstall.exe(Windows)。

8.6.3 重启应用

（1）重新编译应用（非必须，根据补丁要求进行）。

（2）生成二进制配置文件（非必须，根据补丁要求进行）。

tmloadcf **

（3）启动应用。

第 9 章　如何用好全局事务

9.1　什么是全局事务

全局事务是由资源管理器管理和协调的事务，可以跨越多个数据库和进程。事务管理器一般使用 XA 二阶段提交协议与“企业信息系统（EIS）”或数据库进行交互。

也就是当一个事务需要跨越多个数据库时，需要使用全局事务。例如，一个事务中可能更新几个不同的数据库。对数据库的操作发生在系统的各处，但必须全部被提交或回滚。此时，一个数据库对自己内部所做操作的提交不仅依赖本身操作是否成功，还要依赖与全局事务相关的其他数据库的操作是否成功，如果任一数据库的任一操作失败，则参与此事务的所有数据库所做的所有操作都必须回滚。

在一个涉及多个数据库的全局事务中，为保证全局事务的完整性，由交易中间件控制数据库做两阶段提交是必要的。但典型的两阶段提交，对数据库来说事务从开始到结束（提交或回滚）时间相对较长，在事务处理期间数据库使用的资源（如逻辑日志、各种锁），直到事务结束时才会释放。因此，使用典型的两阶段提交相对来说会占用更多的资源，如果网络条件不好，如低速网、网络颠簸频繁，情况会更为严重。

9.2　本地事务的优缺点

本地事务容易使用，但也有明显的缺点：它们不能用于多个事务性资源。例如，使用 JDBC 连接事务管理的代码不能用于全局的 JTA 事务中。另一个缺点是局部事务趋向于侵入式的编程模型。

9.3　Tuxedo 对事务的控制与管理

当客户端连接到 Tuxedo 并创建一个全局事务时，TM（Transaction Manager，事务管理器）就会在公告板（BB）里面创建一个事务，由 TMS 向 GTT（Global Transaction Table，全局事务表，里面包含当前事务的状态信息）中插入一个条目，然后分配一个 GTRID（Global Transaction Identifier，全局事务标识符）来对该事务进行跟踪。

Tuxedo 的事务管理由 TMS 完成，TMS 把各种 RM 接入到 Tuxedo 中的分布式计算中来，并对 RM 中执行的事务进行跟踪和两阶段提交。

Tuxedo 对事务的管理工作主要包括创建 TMS、创建 TLOG、运行时事务的监控和迁移。每一个在 Tuxedo 中用到的 RM，都需要创建一个专用的 TMS，否则无法在 UBBCONFIG

文件中调用。

创建 TMS 的命令为：buildtms。这个命令需要从 RM 文件中读取信息，包括 RM 名、XA Switch 名，以及 XA 支持库。

为了恢复全局事务，TMS 使用 TLOG 来记录事务日志。在每台 Tuxedo 主机上，只需创建一个 TLOG 文件，它就会被这台主机上所有的 TMS 实例共享使用。如果一个全局事务还没有完成，就会在 TLOG 文件中占用一个分页的空间（512KB），事务完成之后，它在 TLOG 中的记录被自动删除。

在全局事务中，如果一个事务在提交前失败，在事务超时以后，TMS 会把它的状态从 TMGACTIVE 改变为 TMGABORTONLY，在 Tuxedo 下一次进行健康检查时，会把它从 GTT 中清除。

另外，当 Tuxedo 检测到只有一个 RM 参与到分布式事务中时，TMS 则会略去第一阶段时的事务征集过程，直接进行事务的提交或者回滚。

9.4　常用事务相关的函数

为了界定全局事务，Tuxedo 除了支持标准的 TX 接口外，还提供了一套自己的事务接口，其中基于 TX 接口的包括以下几种。

1. tpopen()

这个函数被服务进程和 TMS（事务管理器）调用，用于建立和 RM（资源管理器，一般为数据库）的连接。连接信息由服务进程组的 OPENINFO 参数提供。

服务进程和 TMS 在启动时，通常会自动回调 tpsvrinit()函数，tpopen()通常在 tpsvrinit()函数中被调用。连接失败时返回值为–1，并把错误号保存在全局变量 tperrno 中。

2. tpclose()

这个函数在服务进程和 TMS 的析构函数 tpsvrdone(3c)中被隐含调用，用于关闭一个 RM 的连接，关闭信息由进程组的 CLOSEINFO 参数提供。

3. tpbegin()

该函数的功能是开始一个全局事务，并分配一个 GTRID（全局事务标识符）来对它进行跟踪。

4. tpcommit()

该函数的功能是提交一个全局事务，提交成功时返回零，失败时返回–1。

提交失败时，可能把 tperrno 设置为 TPETIME、TPEABORT、TPEPROTO、TPEHAZARD、TPEHEURISTIC 或 TPEINVAL。

- ❑ TPETIME 表示事务已经超时，状态未知，可能是已经提交，也可能是已经回滚。
- ❑ TPEABORT 表示某个 RM 不能提交它的局部事务。

- TPEPROTO 表示协议错误，即调用点不在一个有效的事物上下文中，如事务的提交者不是事务的初始者或者提交的事务根本不存在。
- TPEHAZARD 表示由于某些失败的因素，全局事务已经启发式完成。
- TPEHEURISTIC 表示由于启发式的决策，部分 RM 提交了事务，部分 RM 回滚了事务。
- TPEINVAL 表示函数调用的参数设置不对。

5．tpabort()

回滚一个全局事务。

6．tpsuspend()

该函数功能为挂起一个全局事务。当某些对 RM 的操作不想纳入当前的事务上下文中时，可以在调用点之前先挂起事务，当前事务完成后，再恢复事务。

7．tpresume()

恢复一个被挂起的全局事务。

8．tpscmt()

该函数的功能是设置提交控制参数 TP_COMMIT_CONTROL 的值。

9．tpgetlev()

通过该函数的返回值来判断当前的调用点是否处在全局事务中。如果返回值是 1，表示当前调用点正处在全局事务中，如果是零，表示不在全局事务中。

9.5 数据库连接

9.5.1 TMS 介绍

Tuxedo 事务管理器（TMS）必须跟踪分布式事务处理的整个流程，记录足够的信息以便在任何时候进行提交或回滚，因此 TMS 使用事务日志文件（TLOG）来记录跟踪信息，同时为了区别系统中同时进行的不同事务处理流程，TMS 又为不同的事务处理分配了一个全局事务编号（GTRIDs）。

在事务处理的不同阶段，TMS 将执行不同的动作，见表 9-1。

表 9-1

阶段	TMS 动作
应用程序启动一项事务处理	为事务处理分配一个全局事务编号（GTRIDs）
启动事务处理的进程与其他进程通信	跟踪这些参与事务处理的进程

续表

阶段	TMS 动作
事务处理访问 RM	将相应的 GTRIDs 传递给 RM，这样 RM 就可以监控哪些数据库记录被该事务处理存取
应用程序标记一项事务处理将被提交	按两步提交协议执行事务
应用程序取消事务处理	执行回滚操作
有错误发生	执行回滚操作

9.5.2　XA 模式与 NO-XA 模式

XA 就是 X/Open DTP 定义的交易中间件与数据库之间的接口规范（即接口函数），交易中间件用它来通知数据库事务的开始、结束以及提交、回滚等。XA 接口函数由数据库厂商提供。

NO-XA 应用服务器不需要参与事务管理，只针对单一事务资源，不能跨越多个事务资源。

9.5.3　Tuxedo 与各种数据库的连接

Tuxedo 可以和所有的有标准 XA 接口的 RM 连接，目前几乎所有的关系型数据库和消息队列产品都支持标准的 XA 接口。Tuxedo 和各种数据库相连，都需要配置一个重要的文件 RM。

RM 文件包含所有的资源管理器的入口，它们被 Tuxedo 应用访问，RM 文件在 $TUXDIR/udataobj 目录下。

下面以 Oracle 数据库为例进行介绍。

1．操作系统的准备工作

如果 Tuxedo 连接的数据库不在本地，需要安装 oracle 客户端。

2．Oracle 数据库中的准备工作

Sysadmin 登录数据库，执行如下脚本。

示例 9-1：

```
SQL>@$ORACLE_HOME\rdbms\admin\xaview.sql
```

赋权限给 public 用户。

示例 9-2：

```
SQL>grant select on v$xatrans$ to public with grant option;
SQL>grant select on v$pending xatrans$ to public with grant option;
SQL>GRANT SELECT ON DBA_PENDING_TRANSACTIONS TO Scott;
```

3. .profile 文件的设置，需要设置 ORACLE_HOME 并修改 PATH

示例 9-3：

```
ORACLE_HOME=/u01/app/oracle/product/10.2.1/client
export ORACLE_HOME
PATH=$PATH:$ORACLE_HOME/bin
export PATH
```

4. 修改 RM 文件

如果使用的不是 COBOL（Common business Oriented Language）开发的程序，Oracle_XA 的值不需要改变，否则需要作如下修改。

示例 9-4：

```
Oracle_XA:xaosw:-L${ORACLE_HOME}/lib -L${ORACLE_HOME}/precomp/lib/
cobsqlintf.o -lclntsh
```

5. 创建 tms 文件

在 TUXAPP 目录下创建文件 TMS_ORA10G，Tuxedo 通过 TMS_ORA10g 与 ORACLE 数据库采用 XA 协议进行通信。

示例 9-5：

```
buildtms -o $TUXAPP/TMS_ORA10g -r Oracle_XA
```

6. 修改 UBBCONFIG 文件

在*GROUPS 中添加如下内容。

示例 9-6：

```
OPENINFO="ORACLE_XA:Oracle_XA+Acc=P/scott/scott+sqlNet=ORCL+SesTm=100+
LogDir=.+MaxCur=5" TMSNAME="TMS_ORA10g" TMSCOUNT=2
```

9.6 全局事务的使用规则

全局事务的使用遵守两阶段提交协议，另外在事务控制问题上还有以下几个方面需要注意。

9.6.1 谁发起谁结束

全局事务的发起和结束可以是中间件应用的前台，也可以是后台。在事务的控制上应遵循谁发起事务，谁就结束的原则。

在 Tuxedo 中，事务既可以在前台程序中发起，也可以在后台程序中发起。无论放在

前台还是放在后台都有其优缺点。事务放在前台增加了网络传输的流量，但是可以保证异常情况下前后台操作的一致性，事务放在后台可以减少网络流量，但是对于异常情况下前后台操作的一致性很难保证。

通常采用的是事务放在前台程序中。但是无论放在前台还是后台，都要遵循“谁发起，谁结束”的原则。

9.6.2　不允许嵌套

Tuxedo 不支持嵌套事务处理，即发起者在调用 tpbegin()和 tpabort()或 tpcommit()之间不能再调用 tpbegin()开始一个新的事务处理，也不能再开始一个本地事务。

9.6.3　处理好超时

Tuxedo 应用系统的事务超时控制很重要，不设置超时时间对系统来说可能会引发灾难。影响 Tuxedo 全局事务的超时主要有以下 3 种。

一个是在代码中的 tpbegin()超时（值为其参数）时间 T1，它控制整个事务的完成时间。

第二个为数据库 XA 连接的超时（配置文件中的 open_info 中的 SesTm）时间 T2，它控制同一连接中对于分布式事务锁的等待超时时间。

还有一个为数据库中等待数据库对象分布式事务锁释放的超时时间（_dirstributed_lock_timeout）T3。

这 3 个超时共同作用并且有如下的关系：T1<T2<T3。

9.7　事务挂起的问题

在使用全局事务的情况下，可能会遇到事务挂起的情况，即 SERVER 进程还在，但是无法响应请求。在服务对应的日志文件中总是报“当前的进程已经在一个本地事务中了”的错误，这时只有重启该 SERVER，才能排除故障。其实这是一个事务控制的问题，它产生的根本原因是在开启全局事务前，该 SERVER 执行了一个本地的事务并且没有提交或回滚。在程序代码上表现为如下 3 种原因。

（1）在调用该 SERVER 的某个带 DML 语句的 SERVICE 前没有加 tpbegin。

（2）在调用 tpbegin 后没有判断返回值。

（3）tpbegin 后的 tpcall 服务调用没有判断返回值，在全局事务超时后没有能够及时地退出后续的调用，导致下一个 tpcall 产生了一个本地事务。

此外，还会有一类比较特殊的问题，即 TMS 服务挂起的问题，利用 tmadmin 中的 pq 命令发现 TMS 的队列中有很多请求存在，在这种情况下，一般只能等待其执行完成，情况严重时，则需要调整相应的数据库参数，在 ORACLE 中该参数应该设为 max_commit_propagation_delay>=90000（max_commit_propagation_delay 表示 scn 在 sga 里面刷新的最大时间，单位为 0.01 秒）。

在实际的运行维护中发现，即使把数据库参数调整了，有时还会有 TMS 挂起的故障发生。TMS 之所以挂起是因为 TMS 在等待数据库中 DX（distributed execution lock）独占锁的释放，这种独占锁加锁的对象一般都是 ORACLE 的系统对象，而这种独占锁的产生一般是在全局事务中执行着 DML 语句，当这种 DML 语句执行比较慢时，就会引起 TMS 的锁等待现象。此外，一般的查询语句是不会产生锁的（OPS 的 lm_lock 锁除外），但是如果将查询放入一个全局事务中时，它就会产生一个共享的 DX 锁，如果该 DX 锁在全局事务中的查询的调用是通过 ORACLE 的工具包 DBMS_SQL 来进行，那就会产生一个 DX 的排他锁。

基于这些结果，建议在程序的开发过程中要遵循“能不用 XA 全局事务的情况下就不用，能将查询放到事务之外的就不要放到事务之内，能不用 ORACLE 工具包进行开发的就不要用”的原则。

第 10 章　Tuxedo 性能调优

10.1　目标描述

Tuxedo 管理员最核心的能力是能够对 Tuxedo 应用系统进行性能调优，当然，一些性能的提高可以通过操作系统调优和硬件升级实现，另一方面也可以从调优 Tuxedo 应用环境以及调优 Tuxedo 使用的数据库环境入手。

这里只关注 Tuxedo 环境的调优，假设 DBA 可以调优数据库。

当对 Tuxedo 系统进行性能调优时，要记住少往往更好，意思是：较少的 Tuxedo SERVER 能够比有很多没有被正确利用的 Tuxedo SERVER 使用少的系统开支，提供更好的响应时间。

下面相关例子提供了一个手工的方法来执行调优，当然也可以通过使用 Isis 工具来执行调优（该工具是 Integral Technology Solutions 提供的，具体介绍：www.integral-techsolutions.com）Isis 是一个商业工具，但免费试用版可以从 Integral 网站获得。

10.2　调优独立的 Tuxedo 服务

通常单单通过调优 Tuxedo SERVICE 就可以获得最大的调优 Tuxedo 系统的效果，查看哪些 SERVICE 需要调优，可以通过分析 SERVICE 并且确定哪些 SERVICE 消耗了最多的处理时间。

第一步要确定哪些 SERVICE 实际上运行时间超过其他的，以及哪些 SERVICE 被调用得比其他的更频繁。SERVICE 平均响应时间，乘以被调用次数就是 SERVICE 总共的处理时间，当调优 SERVICE 时，消耗最长的处理时间的 SERVICE 是性能调优的重点。

图 10-1 显示了一个在 Tuxedo 上的计费系统的 SERVICE 的处理器负载情况。从图中可以清楚地看到，有个 Tuxedo SERVICE 占据了很大部分的处理器负载（图中橘黄色部分），这个 SERVICE 消耗了 55%的处理器时间，因此很显然这是一个需要着手性能调优的地方。第二的绿色显示部分的 SERVICE 也需要探查一下，因为它相对于系统中其他 SERVICE 消耗了大量的处理器时间。

图 10-1 是从真实的 Tuxedo 应用中获得的。

一旦 SERVICE 确定，接下来就是集中精力调优该 SERVICE。这里确定核心问题是 SERVICE 中使用了低效的 SQL 语句。该问题代码经过适当的修改，很大程度地降低了分配到该 SERVICE 上的处理器负载，从而节省了为保障客户端性能而带来的昂贵的硬件升级。

图 10-1 实际上是分析原始 TxRPT 数据而获得的。

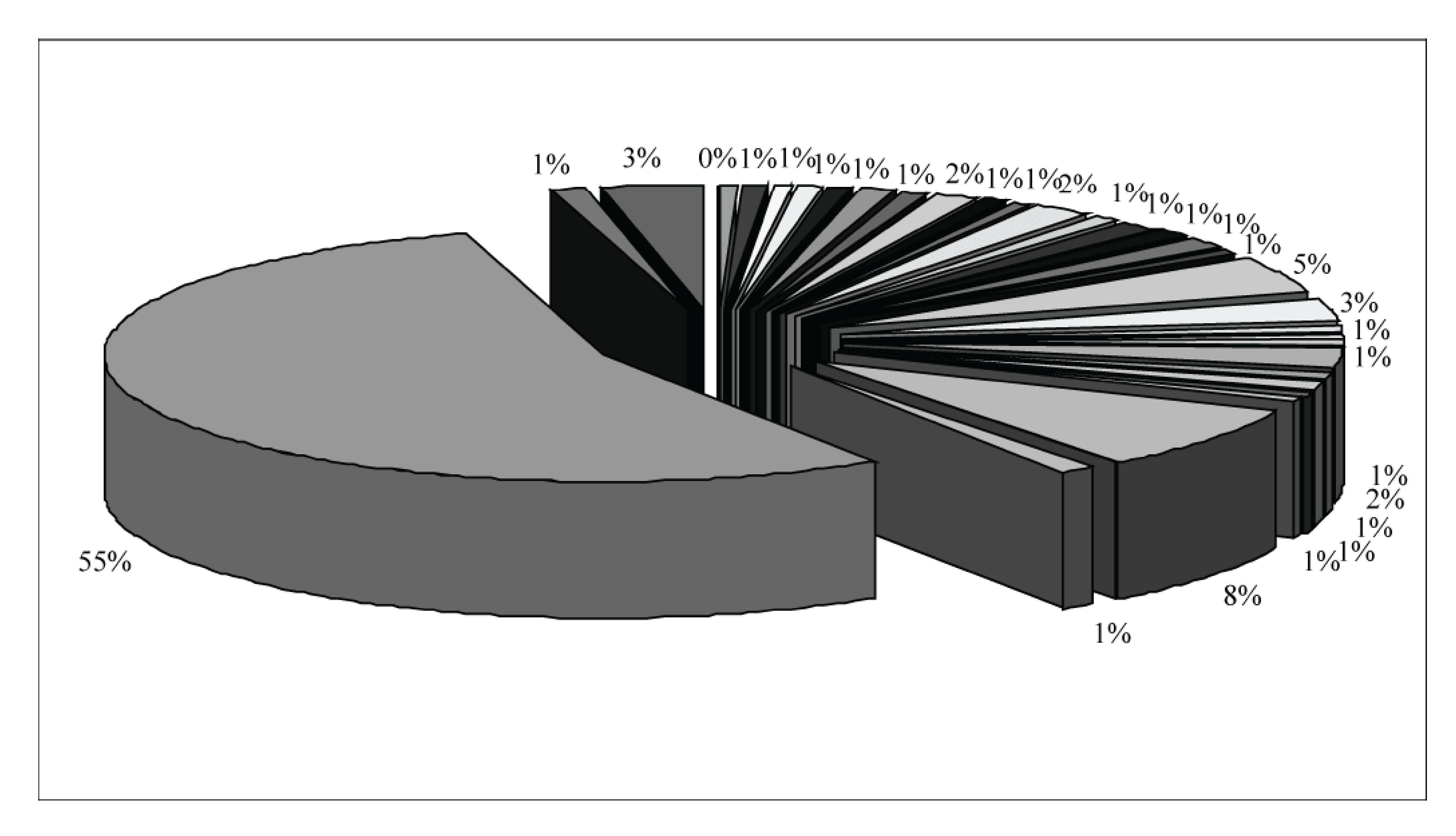
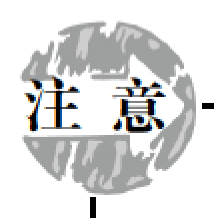


图 10-1 Isis 图显示服务响应时间

TxRPT 数据是 Tuxedo 输出的一种文件格式以提供分析服务响应时间。TxRPT 数据可以通过 Tuxedo SERVER 配置项 CLOPT 的-r 参数打开来获得。当 Tuxedo SERVER 启动并且 SERVER 的-r 打开，将输出 TxRPT 数据到 stderr 文件中，Tuxedo 允许通过 Tuxedo SERVER 配置项 CLOPT 的-e 参数来重定向 stderr 输出文件。

示例 10-1：

```
datasvr SRVGRP=GROUP1 SRVID=10 MIN=2 MAX=2 CLOPT="-A -r -e stdout_simple"
```



-r 和-e 开关是在 CLOPT 的标志“--”之前。

当-r 打开，Tuxedo SERVER 将打印每一次 SERVICE 被调用的情况，输出文件是-e 开关指定的，输出内容如下。

示例 10-2：

```
@CALLDB 3748        1116461950   1548917       1116461955   1553924
```

其中各列分别为 SERVICE 调用的开始时间和日期以及结束的时间和日期，从这些值可以确定服务花费的时间。

Tuxedo 提供了基本的工具来分析 TxRPT 日志的结果，运行 txrpt 工具就是执行一个命令，输入数据文件以及要分析的日期，如下所示。

示例 10-3：

```
txrpt -d 05/20 < stdout_simple
```

将输出类似下面的文件。

示例 10-4：

```
SERVICE SUMMARY REPORT

SVCNAME          16p-17p        TOTALS
                 Num/Avg        Num/Avg
---------------- --------       -------
CALLDB           1081/2.12      1081/2.12
BUSINESSSVC      1082/2.53      1082/2.53
---------------- -------        -------
TOTALS           2163/2.32      2163/2.32
```

在该文件中，可以看到在 Tuxedo 环境中运行着两个 SERVICE，第一个 SERVICE CALLDB 被调用了 1081 次，它的平均响应时间是 2.12 秒，第二个 SERVICE BUSUNESSSVC 被调用了 1082 次，平均响应时间是 2.53 秒。

因为 Tuxedo SERVICE 经常调用其他的 Tuxedo Service，这样就很难确定究竟哪个 SERVICE 是造成性能慢的根本原因。例如，像上面的 txrpt 报告中的 BUSINESSSVC 是运行最缓慢的，就集中精力在这个 SERVICE 上，然而，如果 BUSINESSSVC 要调用 CALLDB，这样造成性能慢的根本原因就是 CALLDB 了，因为相差的 0.41 秒才是 BUSINESSSVC 消耗的，而剩下的时间都是 CALLDB 消耗的。

因此，虽然集中注意力在运行时间长的 SERVICE 上很重要，但要注意 SERVICE 相互调用的依赖关系。

最后，当调优 Tuxedo Service 时，以下是一些容易引起性能问题的地方以及调优步骤的指标。表 10-1 详细地描述了这些常见问题。

表 10-1

常见问题	怎样解决这些问题
低效率的 SQL 访问数据库操作	同 DBA 一起测试调节 SQL 语句，或者调优数据库，让数据库更有效地处理请求的 SQL，如增加索引或者视图来优化相关的请求
同步 Tuxedo 调用	当使用 tpcall API 时，Tuxedo 将阻塞，一直等待调用的结果返回，如果所调用的是一个运行时间较长的 SERVICE，使用 tpacall 以及 tpgetrply 可能更加有效，这样程序就能够继续运行
低效代码	有些时候，性能慢可能是由于低效的代码造成的，使用像代码审查这样基本的技术来找出问题代码，如果还不起效，就应该应用一下高级的技术，如时间戳代码或者代码分析器

为了简化确定性能瓶颈的分析过程，从 www.integral-techsolutions.com 下载 Isis 工具。Isis 简化了导出和管理 TxRPT 文件，并且能够自动地产生帮助，确定造成性能问题的根本原因。

10.3　将相似的 Tuxedo 服务分组到一个 SERVER

当开发一个 Tuxedo 系统时，一个经常犯的错误就是将一些响应时间差别很大的

SERVICE 放到一个 SERVER 中。

例如，假设一个 Tuxedo SERVER 有两个 SERVICE。

SERVICE A 是一个运行时间很短的 SERVICE（平均响应时间<0.1 秒），该 SERVICE 被用来完成一个审计功能，因此它在运行期间将被调用很多次。

SERVICE B 是一个运行时间偏长的 SERVICE（平均响应时间在 15～20 秒），该 SERVICE 基于审计数据生成报告并且将报告输出到公司打印系统上。Service B 是一天被调用一次或者两次，调用次数依据安全部门检查频率而定。

逻辑上这两个 SERVICE 可以属于同一个 SERVER，但是放到一起会引发性能问题。由于 SERVICE A 是一个运行时间很短的 SERVICE，SERVICE B 是一个运行很少的 SERVICE，所以 Tuxedo 管理员决定只运行两个该 SERVER 实例来处理请求。

经过一天天的操作，管理员发现用户间歇性地出现超时和缓慢，这些现象无法解释。

进一步探查发现，在某一个性能负荷的情况下，大量的队列请求出现在等待审计 SERVICE，这和系统性能放缓相符合。然而，管理员还是不能确定为什么会出现性能放缓。

分析 TxRPT 数据可以发现 SERVICE 的平均响应时间以及被调用的频率和时间。通过分析，管理员可以确定造成系统放缓的原因是在一分钟内出现了两个或多个报表服务。这个报表服务将两个 Tuxedo 实例都用来生成报表，但这意味着其他任何审计请求都必须加入请求队列一直等到报表完成。

审计的平均响应时间是少于 0.1 秒，然而当审计请求被放入请求队列等待报表请求完成时，这个响应时间将暴增到差不多是审计时间和报表时间的和。

总共响应时间=15～20 秒+0.1 秒

另外，要考虑这点，很多审计将加入请求队列，并且要逐一运行，当审计每秒可以处理 10 个请求，报表完成要花费 20 秒并且同时出现两个请求时，将发生以下情况。

（1）每个审计 SERVER 实例都将执行报表服务。

（2）当报表请求出现，审计请求继续发来，在 20 秒内将收到 200 个审计请求（如果审计请求是每秒 10 个），这些请求将加入请求队列。

（3）当审计是同步运行时，那么将有 200 个客户端被阻塞，一直等待审计请求通过。

（4）最终报表服务完成了，进程开始处理审计请求，然而两个实例有 200 个请求需要处理，将需要另外的 10 秒来处理完所有请求，才能使系统恢复到正常。

（5）这意味着平常 0.1 秒响应时间的审计将要花费 10～20 秒（取决于请求到来的时间），导致减少了大量潜在的请求进入系统。

上面描述的情形能够通过更好地组织 SERVICE 来解决。很明显上面的例子要考虑将审计和报表服务分开，然而实际应用中怎样确定正确的 SERVICE 组合将会很复杂。

答案很明显，并且能够通过人工的技术或使用 Isis 自动的技术来获得。需要画一个简单的图表，该图包括 4 个象限，两个坐标轴，横轴是 SERVICE 调用的次数，纵轴是 SERVICE 的平均响应时间，并将所有的 SERVICE 标示在图表中。

最终将得到如图 10-2 所示的图表。

当图表绘制完后进行检查图表，并确定有相似的特征的业务功能。

例如，在图 10-3 中标注的部分是有和用户管理相关的 SERVICE 的业务功能，这些 SERVICE 有相似的负载和响应时间并且是服务于一个相似的业务，可以将这些 SERVICE

组合在一个 SERVER 中。

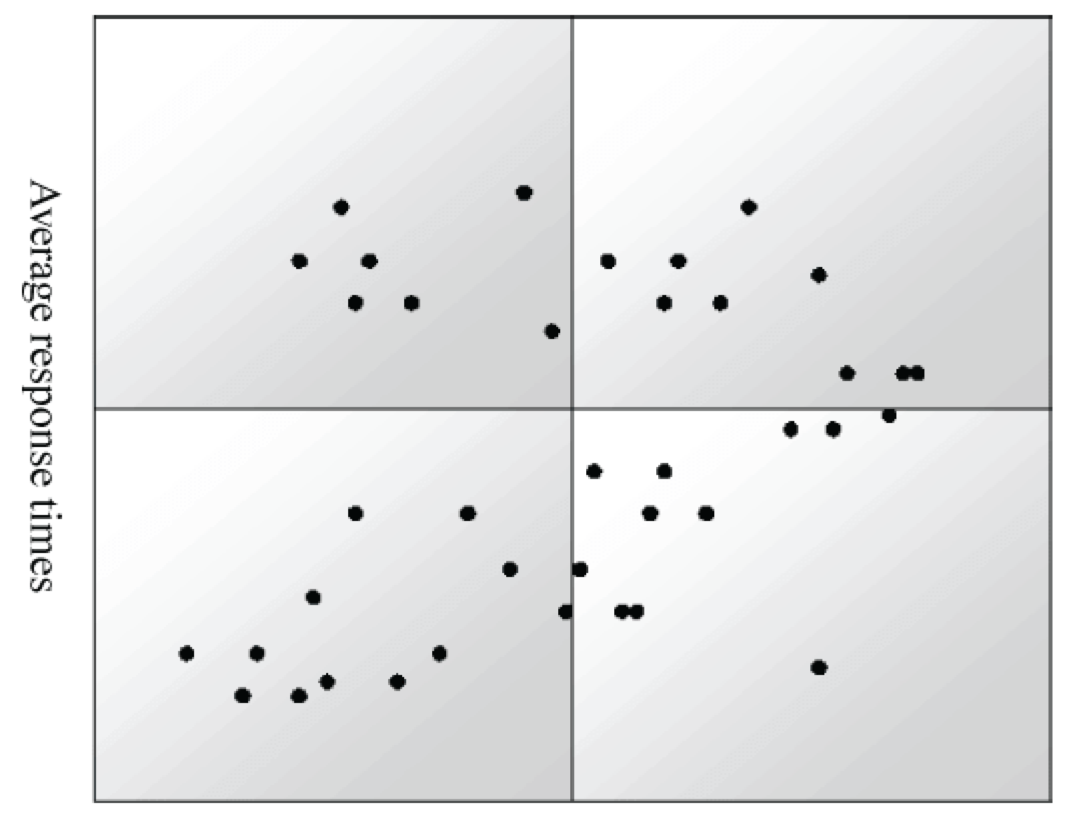


图 10-2

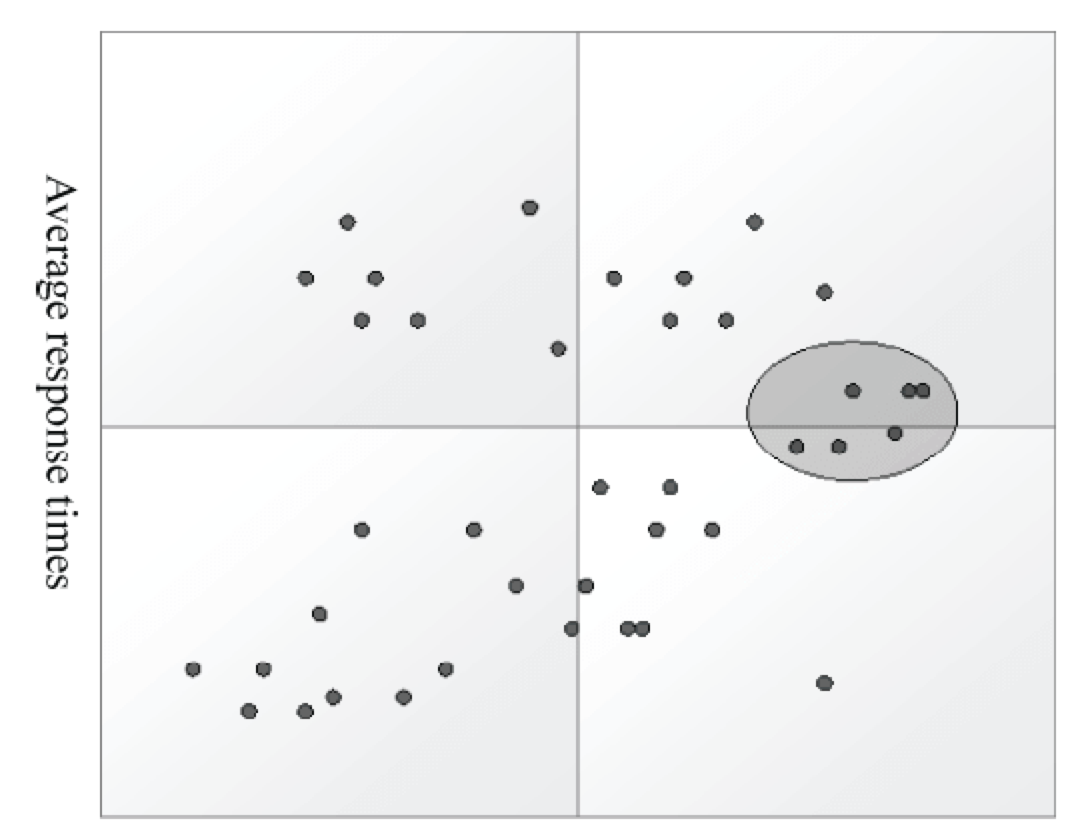


图 10-3

图 10-4 显示了审计和报表服务的点，它们在图表中的逻辑关系是非常不适合的，因此如果放到一个 SERVER 中就不会很好地匹配。

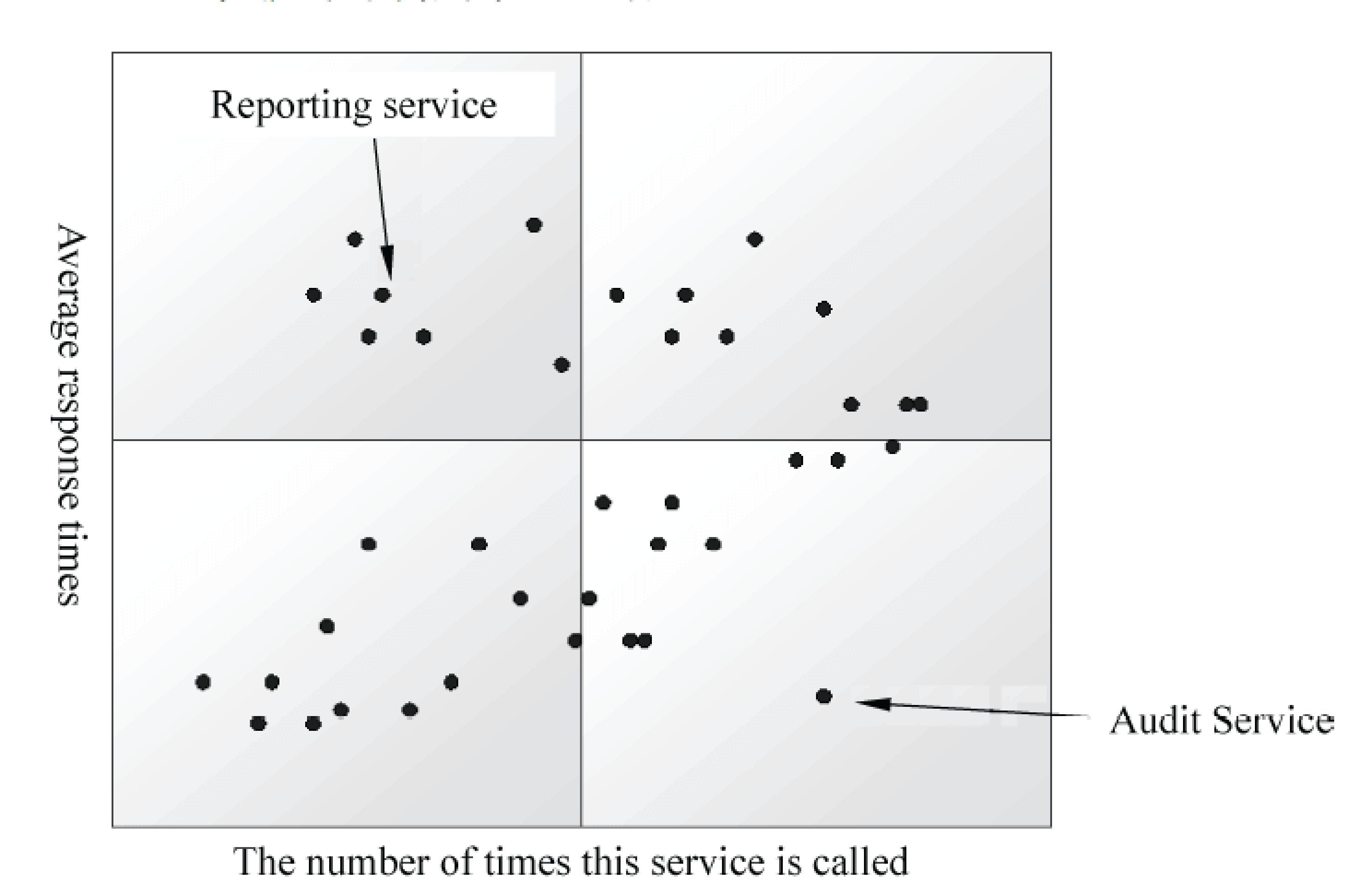


图 10-4

Isis 能够通过获得真实生产数据自动生成上面的图表来帮助 Tuxedo 性能调优。

10.4　调整 SERVER 数量

当系统中的 SERVICE 调试合适并依照负载情况和业务逻辑组合到 SERVER 中后，接下来就是确定 Tuxedo SERVER 启动的合适个数和 SERVER 的合适的队列模型。

首先考虑队列模型，因为它是在这两个问题中比较简单的一个。

Tuxedo 提供了两种可行的队列模型，第一种是一个 SERVER 一个队列，称为 SSSQ（Single Server/Single Queue）模型，另一种是多 SERVER 共同监听一个队列，称为 MSSQ（Multi Server/Single Queue）模型。

当选择使用 SSSQ 还是 MSSQ 模型时，需要考虑以下几个方面。

（1）在 SSSQ 模型中，Tuxedo 将会把请求放到负载最低的 SERVER 请求队列中，当请求信息在队列中时，不能够被其他 SERVER 拿出来，而要一直等待这个 SERVER 处理它。在 MSSQ 模型中，多个 SERVER 使用一个队列，其中任何一个 SERVER 空闲时，请求信息将出队列被处理。

（2）SSSQ 模型在 SERVER 中所有的 SERVICE 有相近的响应时间时将工作顺利。如果一个 SERVER 中 SERVICE 的响应时间不同，使用 MSSQ 模型将更好，因为这样能够避免上面描述的响应快的 SERVICE 被响应慢的 SERVICE 阻塞的情况。

（3）如果一个 MSSQ 中有大量的 SERVER（建议在一个 MSSQ 中不要超过 8 个 SERVER），而服务请求数又很少，将会造成性能放缓，这是因为当一条请求信息到达队列时，所有的 SERVER 都被告知，所有的 SERVER 竞争来获得该信息，当然只有第一个能够获得，所以当队列中有很多 SERVER 时，系统将花费大量时间来唤醒 SERVER 处理请求，而得到的结果却是发现信息已经处理完了。

（4）顾名思义，在一个 MSSQ 中只有一个队列，因为分配给一个队列的内存空间由操作系统固定了，所以可用内存比 SSSQ 少很多。考虑以下情况，如果在一个 SSSQ 模型中有 10 个 SERVER，队列长度为 64KB（一般队列长度），这样总共有 640KB 可用来存放 SERVER 的请求信息，同样的情况，MSSQ 只有 64KB，当 Tuxedo 的队列使用空间超过 80% 时，为保证可靠性，将会把请求信息存放到硬盘上，这样将造成很大的性能影响，因此如果应用的请求信息比较大，应该更多地考虑应用 SSSQ 而不是 MSSQ，因为 SSSQ 能提供更多的可用内存来存放更多的信息。

表 10-2 总结了选择 SSSQ 或 MSSQ 应考虑的因素。

表 10-2

应用背景	消息偏小	消息偏大
SERVER 中的 SERVICE 有相近的负载，并且都是响应时间短以及频繁调用	MSSQ 或者 SSSQ 都能够正常地工作	考虑 SSSQ，因为大量的调用以及偏大消息将可能导致 MSSQ 队列溢出
SERVER 中的 SERVICE 有不同的负载情况或者某些 SERVICE 响应时间很长	当 SERVICE 有不同的负载情况时，MSSQ 工作更好，因为如果一个 SERVER 阻塞在做某些工作，其他的能够处理	这是个棘手的情形，因为不同的负载使用 MSSQ 更好，而请求消息比较大又适合使用 SSSQ，综合两者处理可能提供最好的结果 配置多重 MSSQ 队列通过将 SERVER 分配到不同的组，例如，使用三组 MSSQ，每组中 5 个 SERVER 而不是 15 个 SERVER 使用一个 MSSQ

队列模型确定后，接下来是确定应该运行多少个 Tuxedo SERVER 实例。可以通过 3 个基本的步骤来达到这个目的，结合这 3 个步骤可以得到最佳值。

第一步　使用“-p”开关

Tuxedo 可以通过“-p”开关自动决定运行 SERVER 实例个数，“-p”是 CLOPT 的一个参数。

“-p”开关非常有用，它允许基于实际的负载调整 SERVER 实例个数。因为“-p”开关是被动的，经常稍微落后于真实的负载，因此有时不能及时反应当时需要的配置。

第二步　使用 tmadmin 监控 SERVER 使用情况

当运行在一个 SSSQ 配置上，即使配置了负载均衡，负载在 SERVER 实例中也并不是完全均匀地分布。当 Tuxedo 收到一个请求时，它将发送这个请求到第一个可用的 SERVER 中，如果 SERVER 都不空闲，才发送给负载最少的 SERVER。

如果许多 SERVER 可以接收同一个请求（如在一个 SSSQ 有一个 SERVER 的 10 个实例当前都空闲），Tuxedo 将发送请求到同一个 SERVER，如果第一个 SERVER 繁忙，Tuxedo 将发送请求到第二个 SERVER，以此类推。

久而久之，Tuxedo 的负载均衡算法特征意味着，例如，50%负载将被第一个 SERVER 处理，30%将被第二个 SERVER 处理（由于只有在第一个繁忙的情况下第二个才被调用），15%将被第 3 个 SERVER 处理，5%将被第 4 个处理，等等。

执行 psr（打印 SERVER）命令将显示系统中 SERVER 实例被调用的次数（当运行在一个 MP 环境中记住执行 d-m all 命令），可以给出类似以下的输出。

示例 10-5：

```
> d -m all

all> psr -g GROUP2
Totals for all machines:
Prog Name      Queue Name  Grp Name      ID RqDone Load Done Machine
---------      ----------  --------      -- ------ --------- -------
calldb.exe     calldb      GROUP2        20    10        500  landingbj
calldb.exe     calldb      GROUP2        21     5        250  landingbj
calldb.exe     calldb      GROUP2        22     3        150  landingbj
calldb.exe     calldb      GROUP2        23     0          0  landingbj
calldb.exe     calldb      GROUP2        24     0          0  landingbj
```

在以上例子中可以发现第一个 SERVER（实例 ID20）被调用的次数是最多的，实例 21 和 22 次之，从负载发现 SERVER 实例 23 和 24 没有被调用，这是一个很好的迹象表明这些 SERVER 实例没有必要，可以手动关闭或者配置-p 开关自动关闭。

第三步　分析请求数来确定理想的 Tuxedo 配置

检查平均响应时间和调用频率，考虑 SERVER 中包含了多少 SERVICE，然后结合这些因素确定 Tuxedo 的使用率，通过这些步骤确定理想的 Tuxedo 配置。基于真实的负载来调试 Tuxedo 系统以达到最大产出。

当使用率确定后，接下来给出一个公式用来确定需要多少 SERVER 实例来避免阻塞的关键问题。如果能达到最少的 SERVER 实例而保证无阻塞出现，这就是 Tuxedo 配置的理想情况。

下面给出一个非常简单的例子。

如果已知有一个 SERVER 中包含一个 SERVICE 将花费 1 秒响应时间，便可以得知系统每秒钟能够处理一个事务。

如果已知每小时有 8000 个服务请求（可以通过 TxRPT 文件确定），就能够计算出每秒

需要处理的服务请求个数。

每秒的服务请求数= 8000 / 3600

计算出每秒有 2.22 个事务，所以便可以通过 3 个 SERVER 来处理这些负载。

很显然这是一个简单的例子，而真实的世界中 SERVER 包含多个 SERVICE，SERVICE 被调用的次数，以及响应时间都更加复杂，然而这个相同的准则可以用来获得理想的 Tuxedo 配置。

Isis 提供了内置的报表，该报表能够分析环境中的负载来帮助确定理想的配置，如果使用这个方法来调优 Tuxedo 环境，建议可以考虑 Isis 工具来获得帮助，因为该工具非常有效地简化了分析。

10.5 FML 性能

如果 Tuxedo 应用使用大 FML 缓存可能会发现性能有所下降。下面的建议有助于解决使用 FML 导致的性能问题。

当 FML 缓存块变得越来越大时，将会遇到性能问题，这是由于 FML 在 Tuxedo 缓冲中包装的方式，对 FML 缓存块的包装方式的理解有利于懂得如何有效地处理 FML 结构。

每一个 FML 字段都有一个称为 FIELDID 的数字，如果检查 mkfldhdr32 创建的头文件，会发现定义的每一个字段都会伴随着一个 FIELDID，这个值对考虑怎样包装 Tuxedo FML 缓存块非常重要。

一个 FML 中的字段是按 FIELDID 的顺序储存的，例如，如果在一个 FML 中有 3 个字段（A, B, C）并且 FIELDID 分别是 1、2 和 3，每个字段出现 3 次，那么实际的 FML 如下所示。

示例 10-6：

```
[INDEX]:A1,A2,A3,B1,B2,B3,C1,C2,C3
```

以这种结构形式的问题是当插入字段 A 的第 4 个实例时，实际上需要移动缓存块来为这个字段创建空间，如插入 A4 的步骤如下。

Step 1：为 A4 腾出空间。

示例 10-7：

```
[INDEX]:A1,A2,A3,→ Clear Space for A4 → B1,B2,B3,C1,C2,C3
```

Step 2：插入 A4。

示例 10-8：

```
[INDEX]:A1,A2,A3,A4,B1,B2,B3,C1,C2,C3
```

当处理非常大的缓存块时，以这种方式增加字段将造成系统慢下来的性能问题，这个问题有两种解决方法。第一种方法（也是最简单的）是使用一个字段来储存多个值，例如，如果字段 A、B、C 代表了从数据库取出的列，然后可以用逗号分开 A、B、C 值，最后以一个字段返回（称为 D），这个方法将把 FML 改成如下形式。

Step 1：逗号隔开 A、B、C 然后创建新字段 D。

Step 2：为数据库中取到的每一条数据插入一个 D 字段，FML 如下。

示例 10-9：

```
[INDEX]:D1,D2,D3
```

当取得 D 字段中的数据后，再使用分离算法返回一个字段中的多个值。

虽然这种方法简单，但是增加了缓存结构体的复杂性，并且需要每个收到该缓冲块的客户端懂得怎样提取字段中的内容。

第二种方法比第一种方法复杂，但提供了一种不需要客户端额外去解包的方法。

当插入字段到 FML 缓存块时，如果 FML 储存都按顺序插入，那么插入的性能将大幅改善，原因很简单，因为 Tuxedo 系统不需要为了插入移动字段，例如，如果存在的 FML 如下。

示例 10-10：

```
[INDEX]:A1,A2,A3
```

然后增加字段 B1 将不需要移动任何字段，然而当增加了 B1，如下所示。

示例 10-11：

```
[INDEX]:A1,A2,A3,B1
```

开发人员决定插入 A4，这样 B1 将需要移动，也会造成性能问题。

因此，使用这种方法避免性能问题的关键是在插入字段 B 和 C 之前插入所有的 A，记住正确的字段顺序能够通过查看 FIELDID 得到，这样先插入最小的 FIELDID，以升序插入剩下的字段。

但是，数据通常不是以一个适当的格式返回给开发者来以此种方式插入，例如，每次一个 SQL 查询返回一行数据，要实现以上方式插入 FML，经常需要缓存结果数据并要分开操作来插入 FML 缓存块。更高级的客户端有时封装这些功能到一个分开的 FML 库集合中，以此来开发更高性能的 FML。

10.6　额外的性能参数

以下一些附加的步骤，可能改善 Tuxedo 应用性能。

10.6.1　多个 WSH 连接

当一个工作站客户端连接到一个 Tuxedo 应用，工作站处理程序（WSH）作为一个本地客户端代表工作站客户端来运行。每个 WSH 能够处理很多并行的客户端连接。

WSL 配置项“-x”参数表示每个客户端处理进程能同步接入客户端连接的个数，默认的是 10，这个值必须大于零。

依据通过 WSH 发送的负载情况，以及连接到应用的客户端个数，能够通过多路复用

来调优达到应用的需求。算出复用因子是一个反复的过程，需要使用系统工具分析 WSH 进程的负载情况来确定。

10.6.2 关闭 WSL / WSH 加密

WSL 的“-z”和“-Z”参数定义了将要通过网络的从工作站客户端到工作站处理器的加密级别。

“-z”代表最小的级别，“-Z”代表最高的加密级别，默认为 0。

如果从客户端到工作站进程不需要加密的通信链路，那么推荐检查该值是否为 0 或者该值没有设置。

10.6.3 打开 WSL / WSH 压缩

在 WSL/WSH 上打开压缩能在很大程度上改善 Tuxedo 网络传输的性能，尤其在广域网或者 ADSL/ISDN 方式的连接。

压缩功能的开关是通过 WSL 的“-c”参数来确定的。

“-c”参数决定了工作站客户端和处理进程之间的压缩属性，任何在工作站客户端和处理进程之间的缓存块大于给定值那么就会压缩，默认值是 2147483647，意味着当 buffer 大小在 0 和 2147483647 之间，就不会压缩。如果该值设为 0，意味着不论 buffer 大小都压缩，注意避免这种情况。

10.6.4 机器类型

当 Tuxedo 数据通过网络链路传递时，需要转换 buffer 为机器无关的格式，从而确保接收方的值和发送方的值保持一致。

例如，考虑这样的情况，当一个 Windows 机器以及一个 Solaris 机器被设置在一个网络中，由于两个环境中的数据格式不同，为了让 Tuxedo 系统能够从一个机器到另外一个机器传输一个 buffer，就必须先将数据转换为机器无关的格式，然后再通过网络传输。

这个转换花费相当大的处理时间，如果环境中的所有机器类型都一样，那么就不需要转换。决定是否要转换是通过比较远端的机器 MACHINE TYPE 以及当前机器的 MACHINE TYPE，如果两个值一样，那么 Tuxedo 就不做转换，而是简单地将该信息通过网络传输，这样节省了处理开销。

如果环境中所有的机器类型都一致，那么将*MACHINES 部分的 TYPE 设置为一致，将获得更好的性能。

10.6.5 SPINCOUNT

Tuxedo 无论客户端或服务端何时调用一个服务都要使用公告板（BB），对于公告板的排他性访问，Tuxedo 要通过锁机制进行保护。

参数 SPINCOUNT 表示当锁被占用，而又有新的进程或线程要访问时，它进行多少次拿锁的重试，之后再进入睡眠状态。

在单处理器机器中重试是一个不好的选择，因为这时应该让出 CPU 给占有锁的进程，使它能尽快完成工作。在这种情况下，SPINCOUNT 应该设置为 1。

对于多处理器机器，推荐设置 SPINCOUNT 值在 5000 到 50000 之间。它的设置取决于处理器的类型、个数等多种因素，通常需要通过观察系统吞吐量加以调整。

SPINCOUNT 可以用 TMIB 动态做修改。

10.6.6　去掉授权和审计安全

Tuxedo 版本 7.1 增加了 AAA（authentication, authorization 和 auditing）安全插件功能，这样实现 AAA 插件函数将不需要基于 Tuxedo 管理的安全机制。由于 AAA 接口经常在 Tuxedo 代码中被调用，对于没有使用 AAA 安全插件的应用，就不需要付全额的开销支持 AAA 安全调用。

因此，Tuxedo 版本 8 或以上，RESOURCES 部分的 OPTIONS 参数增加了 NO_AA 选项，NO_AA 选项将避免调用审计和授权的安全函数，而对于需要认证的应用，这个特点不能关闭。

如 NO_AA 启用，以下 SECURITY 参数将受到影响。

NONE、APP_PW 和 USER_AUTH 将继续正常工作——除非没有授权和审计要做。

ACL 和 MANDATORY_ACL 参数将继续正常工作，但只使用默认的 Tuxedo 安全机制。

10.6.7　关闭多线程处理

Tuxedo 版本 7.1 中增加了线程机制，这种架构使得所有 ATMI 调用都必须调用互斥函数以保护敏感状态的信息。此外，库中运用了分层和缓存方案导致了更多的互斥处理。如果应用中没有用到线程，关闭这些线程将大幅度提高性能。

为了关闭多线程处理，Tuxedo 8.0 和后来的版本中实现了 TMNOTHREADS 参数，通过配置可以在不引入新 API 或标志的情况下单个进程可以打开和关闭线程。

如果 TMNOTHREADS 设置为“yes”将避免调用互斥函数。

10.6.8　关闭 XA 事务

对于不使用 XA 事务的应用，为了提高性能，Tuxedo 8.0 以及更新的版本 RESOURCES 部分增加了 NO_XA 标志。

如果设置了 NO_XA，那么将不使用 XA 事务。注意如果 NO_XA 设置了，GROUPS 中设置的 TMS 将失败。

第 4 篇

诊 断 篇

第 11 章　Tuxedo 监控

11.1　监控 Tuxedo 应用的方法

作为一个管理员，必须保证 Tuxedo 正常高效地运行，所以要监控 Tuxedo 的各个方面。

- 资源，如共享内存。
- 活动性，如事件。
- 其他一些问题，如存在必须校正的安全漏洞。

Tuxedo 系统提供了几种方法来帮助用户监控 Tuxedo 系统和应用。

（1）Tuxedo 管理控制台：基于 Web 的图形用户界面，用户可以用它来观察应用程序的行为，并动态地配置其参数。可以显示和更改配置信息，确定系统各组件的状态，并得到执行的情况，如关于项目的统计信息和排队的请求。

（2）命令行方式：可以用一组命令（例如，tmboot，tmadmin，tmshutdown）来启动、关闭、配置和管理应用程序。

（3）日志文件：包含错误和警告消息报告，调试信息，以及对跟踪和解决系统中问题有所帮助的信息。

（4）MIB：使用 MIB，可以编写程序来动态管理监控运行时的应用。

（5）TSAM（Tuxedo System and Application Monitor）。

这些工具帮助用户的应用程序快速有效地对应不断变化的业务需求，提高排除故障情况的能力。还可以帮助管理员管理应用程序的性能和安全。

监控工具如图 11-1 所示。

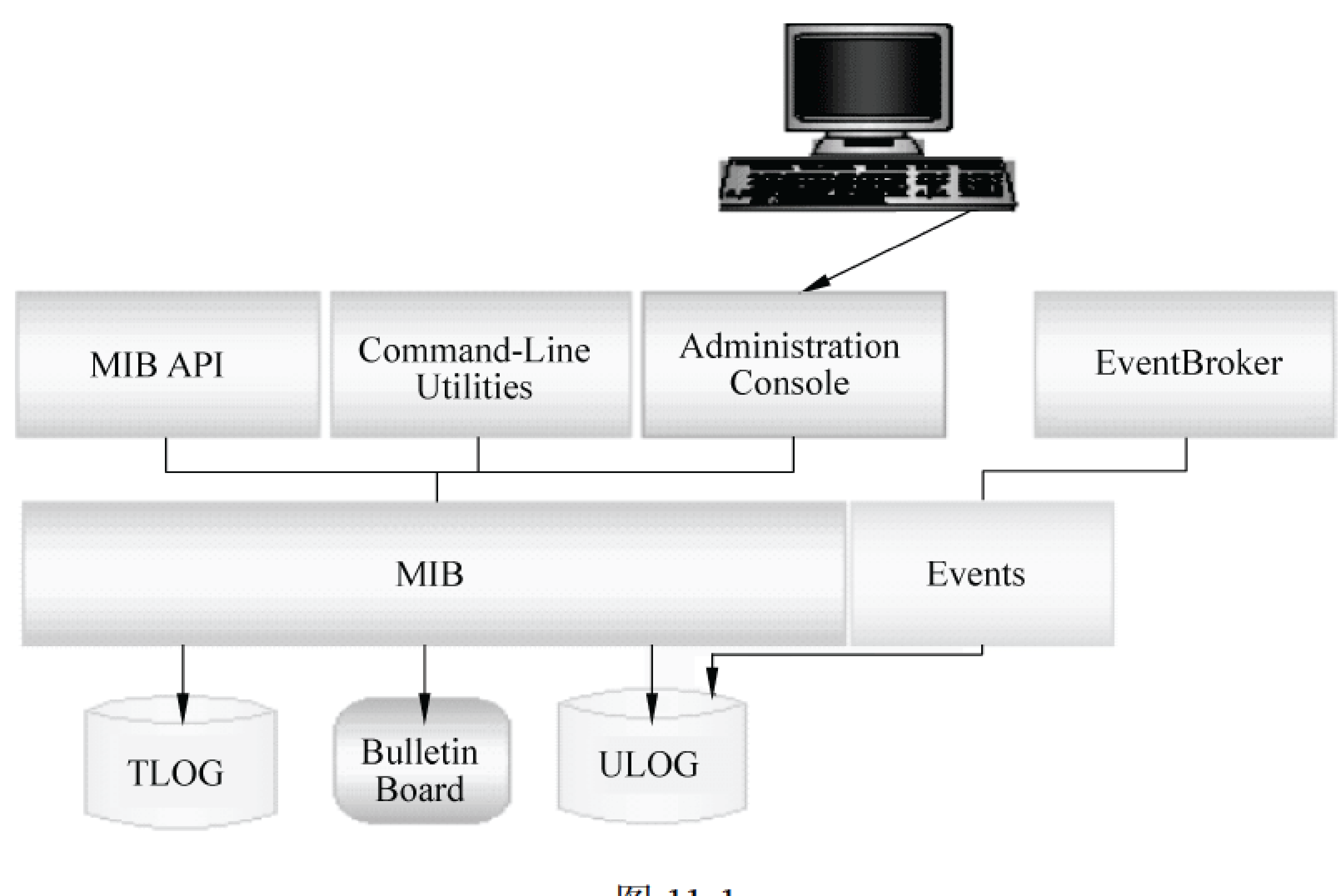


图 11-1

11.2　可以监控的系统和应用数据

Tuxedo 管理员可以用 tmadmin 命令或通过 MIB 来管理监控以下组件。

- ❑ Clients。
- ❑ Conversations。
- ❑ Groups。
- ❑ Message queues。
- ❑ Networks。
- ❑ Servers。
- ❑ Services。
- ❑ CORBA Interfaces。
- ❑ Transactions。

11.3　使用管理控制台监控应用

Tuxedo 的管理控制台是一个图形用户界面，底层通过 MIB 查询修改系统配置、监控应用程序。它是通过 Web 浏览器来访问的。任何一个管理员通过浏览器可以监控 Tuxedo 的应用。

11.4　使用命令行方式监控

在 Tuxedo 启动的情况下，可以用 tmadmin 命令对 Tuxedo 应用进行管理和监控。常用的子命令请参见第 8 章相关内容。

11.5　使用 EventBroker 监视应用程序

首先明确一下基本概念：事件代理（EventBroker），它其实提供了这样一种通信方式，即一些进程可以发布消息给订阅了这些消息的进程或客户端，例如，服务可以在价格发生变化时发布信息，所有连接系统的客户端都可以收到信息。

事件代理只能执行一些预先定义的动作，这些动作包括以下内容。

- ❑ 调用服务（tpacall）。
- ❑ 将请求送入一个可靠队列（tpenqueue）。
- ❑ 用户通知（tpnotify）。
- ❑ 记录日志到 ULOG.mmddyy。
- ❑ 执行系统调用。

主要有以下两类事件。

- **系统事件**　Tuxedo 系统内部状态变化，如网络故障等。
- **用户定义事件**　应用中与业务流程有关的事件。

11.5.1 相关 API 介绍

（1）发布事件用的 ATMI API 是 tppost()。

示例 11-1：

```
int tppost(char *eventname, char *data, long len, long flags)
```

char *eventname：事件的名字。

char *data：伴随事件发布的数据。

long len：数据长度。

flags：可以是[TPNOTIME|TPSIGRSTRT|TPNOBLOCK]。

（2）订阅和取消订阅事件的 ATMI API 是 tpsubscribe()/tpunsubscribe()。

示例 11-2：

```
long tpsubscribe(char *eventexpr, char *filter, TPEVCTL *ctl, long flags)
```

char *eventexpr：表示事件的字符串，可以含有通配符*。

char *filter：一个布尔表达式，决定是否触发相应的动作。

TPEVCTL ctl：描述事件代理将进行的动作。

Flags：可以是[TPNOTIME|TPSIGRSTRT|TPNOBLOCK]。

以下函数返回订阅句柄，–1 表示失败。

示例 11-3：

```
int tpunsubscribe(long subscriptn, long flags)
```

long subscriptn：tpsubscribe()返回的句柄。

flags：可以是[TPNOTIME|TPSIGRSTRT|TPNOBLOCK]。

（3）TPEVCTL 结构的定义如下。

示例 11-4：

```
 struct TPEVCTL
  {
      long flags;
      char name1[32];
      char name2[32];
      TPQCTL qctl;
  }
```

（4）订阅一个事件，指定事件发生时触发一个服务请求的方法如下。

示例 11-5：

```
ctl->flags=TPEVSERVICE;
```

```
strcpy(ctl->name1,"SERVICENAME");
```

（5）订阅一个事件，指定事件发生时把消息存入队列的方法如下。

示例 11-6：

```
ctl->flags=TPEVQUEUE;
strcpy(ctl->name1,"QSPACENAME");
strcpy(ctl->name2,"QUEUENAME");
ctl->qctl.flags=TPQREPLYQ;
strcpy(ctl->qctl.replyqueue,"RPLYQ");
```

11.5.2 相关例子参考

下面给出一个使用 tpsubscribe()和 tpunsubscribe()的例子。

示例 11-7：

```
#include <atmi.h>
static long sub serv;
int tpsvrinit(int argc, char **argv)
{
  TPEVCTL evctl;
  /* 连接 Tuxedo 系统—tpinit() */
  evctl.flags=TPEVSERVICE;
  strcpy(evctl.name1,"BANK MANAGER");
  sub serv=tpsubscribe("BANK TLR WITHDRAWAL",
    "AMOUNT>300.00",&evctl,TPSIGRSTRT);
  if (sub serv == -1)
  return(-1);
}

int tpsvrdone()
{
  if (tpunsubscribe(sub serv,TPSIGRSTRT)== -1)
  {
    printf("Error unsubscribing to service event\n");
  }
}
```

该服务器在初始化时订阅事件，指定事件发生时调用服务 BANK_MANAGER，事件是 BANK_TLR_WITHDRAWAL，仅当 AMOUNT 域的值大于 300.00 时触发。

服务器在终止时用 tpunsubscribe()取消订阅。

下面是使用 tppost()的例子。

示例 11-8：

```
void WITHDRAWAL(TPSVCINFO *transb)
{
    ……
if (amt > 300.00)
```

```
      {
        if ( (Fchg(transf,EVENT NAME,0,"BANK TLR WITHDRAWAL",0)<0) ||
         (Fchg(transf, EVENT TIME,0,gettime(),0)<0 )||
         (Fchg(transf, AMOUNT,0,(char *)&amt,0)<0) ))
       {
         sprintf(emsg,"Fchg(event flds) failed :%s",Fstrerror(Ferror));
       }
       else if( tppost("BANK_TLR_WITHDRAWAL",(char *)transf,0L,
         TPNOTRAN|TPSIGRSTRT)<0)
       {
          if (tperrno!=TPENOENT)
          {
           sprintf(emsg,"tppost() failed :%s",tpstrerror(tperrno));
          }
      }
      if ( strcmp(emsg,"")!=0)
      {
        userlog("WARN:Event BANK TLR WITHDRAWAL not posted:%s",emsg);
        strcpy(emsg,"");
      }
    }
  }
```

WITHDRAWAL 服务在金额大于 300.00 时，给 FML 域赋值，并调用 tppost()发布事件。而相关的事件代理服务器要配置在 ubbcofig 中。

示例 11-9：

```
*SERVERS
TMSYSEVT    SRVGRP=EVTGRP1  SRVID=100
            RESTART=Y MAXGEN=5 GRACE=3600
            CLOPT="-A  -- -f tmsysevt.dat"
TMUSREVT    SRVGRP=EVTGRP1  SRVID=150
            RESTART=Y MAXGEN=5 GRACE=3600
            CLOPT="-A  -- -f tmusrevt.dat"
TMUSREVT    SRVGRP=EVTGRP1  SRVID=200
            RESTART=Y MAXGEN=5 GRACE=3600
            CLOPT="-A  -- -S -p 120"
```

TMSYSEVT：系统事件代理服务进程。

TMUSREVT：用户事件代理服务进程。

-f：指定了包含订阅信息的数据文件。

11.6 使用 MIB 监视应用程序

Tuxedo 提供了一套可编程的管理员 API 接口（Management Information Bases，MIB）。

通过 MIB，可以方便地监控 Tuxedo 运行时的所有系统信息，如 SERVER 和 SERVICE 的运行状况、SERVER 队列的情况、客户端的使用情况、域间通信的连接情况、系统的资源配置等，所有的信息都可以通过 MIB 的 API 来获取，或者动态地修改系统配置。

例如，对于某些关键的 SERVER 或者 SERVICE，想监控它在一段时间内的被调用情况，服务是否出现异常或者是被挂起，SERVER 对应的消息队列是否出现了堵塞，或者是队列空间紧张；当 SERVER 出现请求反应较慢需要增加进程时，也可以通过 MIB 动态地增加进程个数。

例如，对于消息队列，需要监控消息队列当前的请求总数，队列长度是否达到了危险程度需要告警。

例如，对于客户端使用者来说，需要监控有哪些客户端调用超时或者是调用已经被系统给挂起的情况，某些客户端调用出现异常会导致系统性能下降，可以自动先将这些客户端请求给撤销或者杀掉，再根据客户端请求的全过程来找出问题。

例如，如果系统和其他外围系统存在着连接，或者是分布式的 Tuxedo 系统，需要监控系统和其他系统的域间通信连接是否正常，远端的系统是否还在正常运行。

用一句话来概括：Tuxedo 所有的应用都可以通过 MIB 进行实时监控。

MIB 接口规定了系统管理员、系统操作员和其他用户，三者具有不同的操作权限。

MIB 接口将系统资源信息划分为 3 种类型，分别为 Classes、Attributes 和 States。

- Classes 指资源的分组，如 SERVER、SERVICE、CLIENT、QUEUE、MSG 等。
- Attributes 指 Classes 对应的属性，如 SERVER 的属性有 SERVERNAME、SRVGRP、SRVID 和运行时的参数值等属性。
- States 指 Classes 在运行时的状态，以及在运行时可以更改的属性。

MIB 的操作类（OPERATION）分为 GET 和 SET 两种类型，即指查询和设置两种操作。

Classes 的类型有 T_MACHINE、T_GROUP、T_SERVER 、T_SERVICE 、T_SVCGRP、T_QUEUE、T_MSG T_DOMAIN、T_CLIENT 等。

用 MIB 的 API 查询系统资源的编程比较简单，与客户端调用 SERVICE 方法一样，都是用 tpcall()或者 tpacall()，只是服务名为.TMIB。输入输出的缓冲区类型为 FML32。在调用之前需要指定几个查询参数 TA_OPERATION 和 TA_CLASS、TA_FLAGS，以及 Classes 对应的参数，例如，在查询 T_SERVER 这个 Classes 时，想根据 LMID、SERVERNAME、SRVGRP、SRVID、RQADDR 等参数来查询 SERVER，那么就在输入缓冲区中输入对应的参数值。

下面以 T_SERVER 这个 Classes 为例子来作个大致介绍。

T_SERVER 就是在 Tuxedo 配置文件的*SERVERS 节中所定义的 SERVER。用 MIB 来查询 SERVER 在运行状态下的数据，可以获取 SERVER 的配置参数值，以及 SERVER 运行时的其他动态参数值，例如，进程 ID（操作系统的 PID），请求消息队列的 ID，回复消息队列的 ID，队列名称，SERVER 当前被请求的服务，SERVER 被调用的次数、已经完成的调用次数，还没有完成的请求数，等等。

代码如下。

示例 11-10：

```
obuf = (FBFR32 *)tpalloc("FML32",NULL,8000);
```

```
    Finit32(obuf, (FLDLEN32)Fsizeof32(obuf));

    Fchg32(obuf, TA_OPERATION, 0, "GET", 0);
    Fchg32(obuf, TA_CLASS, 0, "T_SERVER", 0);
    flags = MIB_LOCAL;
    Fchg32( obuf, TA_FLAGS, 0, (char *)&flags, 0);
    //设置想要查询的参数，如服务名、服务 ID、服务队列等
    if(strlen(t_SrvName)>0) Fchg32(obuf, TA_SERVERNAME, 0,t_SrvName, 0);
    if(strlen(t_SrvGrp)>0)  Fchg32(obuf, TA_SRVGRP, 0,t_SrvGrp, 0);
    ret=tpcall(".TMIB",(char *)obuf,0,(char **)&obuf,&sendlen,TPNOTRAN);
if(ret == -1)
    {
     userlog("获取系统服务的状态失败:%s",tpstrerror(tperrno));
     tpfree((char *)obuf);
    //进行错误处理并释放资源退出
    }
    i=Foccur32(obuf,TA_SERVERNAME);
    for(j=0;j<i;j++)
    {
       Fgets32(obuf,TA_SERVERNAME,j,t_ServerName);
       Fgets32(obuf,TA_SRVID,j,t_SrvID);
       Fgets32(obuf,TA_STATE,j,t_STATE);
       Fgets32(obuf,TA_SRVGRP,j,t_SrvGrp);
       Fgets32(obuf,TA_RQADDR,j,t_RQADDR);       //SERVER 队列名称
     Fgets32(obuf,TA_TOTWORKL,j,t_TOTWORKL);    //工作量。TA_TOTREQC*负载因
                                                  子=TA_TOTWORKL


       ..........................
       //根据返回的数据进行其他操作
       ..............................
   }
```

从上面代码可以看到，用户可以获取到 SERVER 在运行状态下的所有信息。

例如，假设有一个服务发生异常，那么就可以根据获取的服务状态，来自动做一些处理。如果 SERVER 包含多个 SERVICE，那么可以监控到某个 SERVICE 是否因为被调用次数过多而且影响到 SERVER 中其他的 SERVICE，如果是这样就要考虑将这个 SERVICE 移到其他 SERVER 或者是单独成为一个 SERVER。查询 SERVER 对应的请求消息队列，如果排队过长，则需要自动进行处理或者是告警。

如果想在运行状态时动态地修改 SERVER 的配置参数，那么可以将 TA_OPERATION 改为 SET，将需要修改的参数输入缓冲区，调用 MIB 的接口即可。

修改某些参数有时需要将 SERVER 的状态改为 INActive，相当于 tmshutdown-s SERVER，然后才能修改。

11.7　使用日志文件来监控

11.7.1　Tuxedo 日志的分类

Tuxedo 系统提供了两种日志文件，分别是 Transaction Log (TLOG)和 User Log (ULOG)，可以帮助管理员快速准确地发现问题。

- Transaction Log (TLOG)——一个二进制的文件，被事件管理器（TMS）使用，不能用文本编辑器读取，可以用 tmadmin 管理命令查看全局事务的状态。
- User Log (ULOG)——当应用程序启动时同时 Tuxedo 系统产生一个 ULOG 消息日志，记录 Tuxedo 系统输出的信息。用户也可以通过编程输出信息到这个文件。

11.7.2　Tuxedo 事务日志

1．什么是 Transaction Log (TLOG)

TLOG 日志用来跟踪全局事务的两个阶段提交，日志中的记录是用来保证在网络故障或者是机器崩溃时，仍能正常完成全局事务。

对于跨域的全局事务，还需要域日志 DMTLOG 来共同保证事务的完整性。

2．Tuxedo 的事务管理

Tuxedo 的事务管理工作包括创建 TMS、创建 TLOG、运行时事务的迁移和监控。

3．创建事务管理器

系统管理员必须为每一种用到的资源管理器（RM）创建专用的事务管理器（TMS），并且在 UBBCONFIG 文件中，配置使用。Tuxedo 系统提供了 buildtms 命令来创建 TMS，该命令需要从一个名为 RM 的文件中去读取 RM 信息，包括 RM 名、XA switch 名，以及 XA 支持库。

RM 位于 Tuxedo 的 udataobj 目录下，它保存着系统可能用到的所有 RM 信息，每行格式如下。

示例 11-11：

```
RM_NAME:XA_SWITCH_NAME:XA_LIBS
```

其中，UNIX 平台使用“：”分隔，Windows 平台使用“；”分隔。

RM_NAME 是 RM 在 Tuxedo 系统中的标识名，XA_SWITCH_NAME 是 RM 的 xa_switch_t 结构名，XA_LIBS 是 RM 的接口库。XA Switch 名和 XA 接口库由 RM 供应商提供。修改完这些，就可以使用 buildtms 来创建 TMS。Buildtms 有两个选项，“-r”指定 RM 名，用于引用 RM 文件中的一个条目，“-o”用于指定生成的 TMS 名称和存放路径。

建议把 TMS 放在 Tuxedo 的 bin 目录下，以方便重复使用。

4. 创建事务日志

TMS 使用 TLOG 来记录事务日志，以便恢复或回滚全局事务。每台 Tuxedo 主机上只需要创建一个 TLOG 文件。它会被这台主机上的所有 TMS 实例共享使用，如果一个全局事务还没有完成，那么它在 TLOG 中将占用一个分页的空间，全局事务完成后，它在 TLOG 中的记录将会被删除。

创建 TLOG 时先进入 tmadmin 界面，用 crdl 命令创建设备文件，命令格式如下。

示例 11-12：

```
crdl -b blocks -z config
```

（1）-b blocks 为设备文件的大小，以块（block）为单位。

（2）-z config 为设备文件名，应和配置文件中 TLOGDEVICE 相同。

然后执行 crlog 命令创建 TLOG，命令格式如下。

示例 11-13：

```
crlog -m machine
```

其中-m machine 为机器逻辑名。

如果用到跨域的全局事务，则必须为每个域网关组创建一个 DMTLOG 日志。该 DMTLOG 文件被定义在 DMCONFIG 文件的*DM_LOCAL 段。

示例 11-14：

```
DMTLOGDEV=string
```

其中 string 是日志设备的名称。

DMTLOG 还有两个可选参数。

（1）DMTLOGNAME=identifier 用来指定设备上的 TLOG 名。

（2）DMTLOGSIZE=numeric 用来指定 TLOG 的大小。

如果在启动一个 domain 网关组时没有创建 DMTLOG，那么网关服务器会自动创建这个日志。

5. 迁移事务日志

当主机出现故障需要更换，或者是管理员需要对运行 Tuxedo 的系统进行升级时，管理员可以把正在执行的应用程序迁移到另一台主机。

事务日志是迁移中的重要部分，迁移步骤如下。

（1）停止所有可能向 TLOG 中写日志的进程。

（2）执行 migrategroup 把所有部署在源主机上的进程迁移到备份机。

（3）执行 dumptlog 命令把事务日志转储在一个 ASCII 文件的 logfile 中。

在目标主机上执行的命令如下。

（1）从源主机上复制 logfile，执行 loadtlog 命令从 logfile 中加载事务日志。

（2）执行 logstart，命令强制对事务日志做热恢复。

（3）执行 boot 命令启动所有迁移过来的进程组。

6. 处理事务日志

在 Tuxedo 应用程序中，造成事务失败的主要原因有发起事务的客户机异常终止、TMS 或服务进程异常终止、通信网络中断、Tuxedo 系统出现错误和 RM 响应超时等。事务失败可能发生在提交前、2PC 的第一个阶段（预提交事务阶段）和 2PC 的第二个阶段。在提交前和预提交过程中失败的事务可以由 Tuxedo 自动恢复，而在 2PC 的第二个阶段中失败的事务需要在管理员的协助下进行恢复。

如果一个事务在提交前就失败了，那么在事务超时后，TMS 会把它的状态从 TMGACTIVE 改变为 TMGABORTONLY，在 Tuxedo 下一次进行健康检查时，将它从全局事务表（GTT）中清除。如果一个事务在预提交过程中失败了，那么 tpcommit()将会返回–1，并且把 tperrno 设置为 TPEABORT。在事务超时以后，TMS 会把它的状态从 TMGCOMMCALLED 改变为 TMGABORTED，在 Tuxedo 下一次进行健康检查时把它从 GTT 中清除。

如果一个事务在 2PC 的第二阶段失败了，tpcommit()返回–1，tperrno 设置为 TPEHAZARD 或 TPEHEURISTIC。TPEHAZARD 表示事务已经由 RM 启发式完成，但是状态未知；TPEHEURISTIC 表示事务已经由 RM 启发式完成，但是一部分 RM 提交了事务分支，另一部分回滚了事务分支。

在一个事务提交以前，如果因为 TMS 异常终止而导致事务不能正常结束，那么管理员可以通过执行 tmadmin 提供的管理命令来结束它。Tmadmin 提供了 printtrans（pt）、aborttrans（abort）和 committans（commit）来结束没有完成的事务。

7. 查看 TLOG

Tuxedo 日志可以帮助管理员发现和分析系统和应用失败的原因。TLOG 是一个二进制文件，它包含全局事务交易过程的信息。要查看 TLOG 必须先将 TLOG 转换为文本文件，Tuxedo tmadmin 提供了双向转换的命令。

（1）dumptlog (dl)把二进制 TLOG 导出到一个文本文件。

（2）loadtlog 把 dumptlog 导出的文本加载到 TLOG 二进制文件中。

11.7.3 Tuxedo 用户日志

1. 什么是 User Log（ULOG）

User Log 是 Tuxedo 系统运行日志文件，包括 error messages，warning messages，information messages 和 debugging messages。应用客户端和服务器通常也会写这个文件。

每天会产生一个新的 ULOG 日志，并且在每台机器上有不同的 ULOG。

当一个共享文件系统被使用时，多个机器就可以共享一个 ULOG。

ULOG 提供了一个系统管理记录，包含 Tuxedo 系统和应用失败的原因。可以用文本编辑器查看 ULOG 文件。

2. 查看 ULOG

下面来看一个 ULOG 的例子，比如，错误消息是 LIBTUX_CAT:358，问题的原因是没有足够的 UNIX 系统信号量来启动应用程序。

示例 11-15：

```
151550.landingbj!BBL.28041.1.0: LIBTUX_CAT:262: std main starting
151550.landingbj!BBL.28041.1.0:  LIBTUX_CAT:358:  reached  UNIX  limit  on
semaphore ids
151550.landingbj!BBL.28041.1.0: LIBTUX CAT:248: fatal: system init function..
151550.landingbj!BBL.28040.1.0: CMDTUX CAT:825: Process BBL at SITE1 failed ...
151550. landingbj!BBL.28040.1.0: WARNING: No BBL available on site SITE1.
Will not attempt to boot server processes on that site
```

一个 ULOG 消息是由标签和正文组成的。

标签由下面几部分组成。

（1）一个六位数字（hhmmss）表示一天中的时分秒。

（2）机器的名字（在 Linux 中 hostname-n 显示的机器名）。

（3）进程的名称和标识（进程 ID、线程 ID、上下文 ID）。

如果当前处于全局事务中，则还包括一个事务 ID。

正文由下面几部分组成。

（1）Tuxedo 消息目录的名字。

（2）Tuxedo 消息编号。

（3）Tuxedo 系统消息。

另外，关于 Tuxedo 域连接的管理监控，可以通过 dmadmin 命令或 MIB 完成。dmadmin 常用的子命令请参见第 8 章相关内容。

第 12 章　服务 core dump 分析

12.1　什么是服务 core dump 文件

core dump 是一个在特定时间的内存的快照，一般用作调试非正常结束的程序。

在以下几种情形下，core dump 对调试有帮助。

首先，core dump 保存了出错现场的信息，可以供以后分析或者与其他错误作比较。

其次，某些操作系统是不支持将调试程序附加到运行的进程上的，这时 core dump 可以保留内存状态。

在很多操作系统中，程序中的一个致命的错误将自动触发生成 core dump，因此，生成 core dump 意味着有一个致命的错误。

12.2　什么情况可以导致 core dump 文件生成

通常，core dump 的生成是由于应用开发的错误，但也可能是其他原因。例如，操作系统错误；使用 C 或者汇编语言中非法的内存指针，就能造成 core dump；其他的原因，像在 C++中引用的对象没有创建，抛出的异常没有捕获；等等。

12.3　服务器 core dump 文件探查

当 core dump 发生时，一个名为 core 的文件将生成。在 UNIX 机器上通常是在可执行程序运行的路径下（如$APPDIR/bin），而在 Windows 机器上将生成在路径 c:\Documents and Settings\All Users\Documents\DrWatson 下。

以下是分析 core dump 的 3 个主要的步骤。

（1）核实应用宕掉时生成了 core dump。依赖于环境的配置，core dump 可能不会生成。

（2）找到 core 文件，应用此文件探查出错的原因。

① 运用 core 文件诊断工具来获得详情。

② 确定是什么地方导致了错误（操作系统限制，内存指针，驱动错误，操作系统错误，等等）。

（3）探查错误的具体原因。

① 可能需要对应用的代码有详细的了解。

② 可能需要额外的调试跟踪。

③ 排除法和写测试程序也可以确定问题所在。

下面将详细分析每个步骤，注意，为了从 core dump 文件中获得更详细的信息，编译器的“-g”开关需要打开，这样就建立了调试所需的符号信息。

示例 12-1：

```
CFLAGS=-g
CC=/bin/cc
```

然而，打开“-g”开关会使生成的执行程序变大，因此，通常情况下是用在确实需要调试的程序上。

Windows 系统中打开调试功能根据编译器的不同，设置也不同，Microsoft Visual Studio 编译器中，要使用额外调试功能时需要选上“Generate Debug Info”。

12.3.1 检查系统环境以保证 core dump 生成

依赖于环境的设置，有时 core 文件不会生成。为确保当服务出现错误时能够生成 core 文件，需要检查和改变某些设置。

- 检查系统以及用户的 core 文件的大小限制，如通过命令 ulimit。该命令可以查看或设置进程创建的 core 文件的大小限制，如果设置偏小将阻碍应用生成 core 文件。
- 在 Solaris 系统中：检查 core dump 大小的设置，i.e. 在/etc/system 中如果设置了 sys:coredumpsize=0，则不会生成 core 文件。
- 在 Linux 系统中：core dump 默认是关闭的，检查在 /etc/security 下的 limits.conf 文件中 core 的设置。
- 在 HP-UX 系统中：检查用户进程数据段大小，i.e.设置 maxdsiz>134MB。
- 检查用户级的磁盘配额的限制。

另外，通过使用 gcore 工具，可以为指定的进程生成 core 文件。该命令的语法如下。

示例 12-2：

```
gcore <PID>
```

12.3.2 保存 core 文件

每次程序出现严重错误时，将 core 数据写入 core 文件，生成的目录通常是在程序执行的路径下。如果是在 Tuxedo 下，所有的 Tuxedo 服务进程的执行目录都是同一个，所以当 Tuxedo 服务进程出现严重错误时，生成的 core 文件将覆盖前面生成的 core 文件。在一个系统中，如果发生多次 core dump，将很难捕捉到准确的 core 文件。

这个问题的一个可行的解决方案就是利用 Tuxedo 的自动重启功能，在启动脚本中指定将 core 文件复制或移动到另一个目录下，并附上 GRPNO、SRVID 和时间戳。启动脚本是由 UBBCONFIG 中 SERVER 的 RCMD 参数设置的。

这项技术用起来并不是万无一失的，由于 Tuxedo 系统要花费一定时间来识别 SERVER 异常终止以及重新启动，重启的脚本只有在 SERVER 启动时才能运行，如果恰巧这个时候

另外一个 SERVER 也 core dump 了，这样脚本将拷贝被覆盖的 core 文件。所以这项技术也是有局限性的。

12.3.3　找到 core 文件并使用其探测错误成因

有以下几种方法来探查 core dump 错误。

（1）应用日志文件（如 JVM 日志文件，应用调试日志，在 stdout 或 stderr 中的堆栈跟踪），分析这些日志文件通常比分析 core dump 文件更简单。

（2）使用堆栈跟踪工具分析 core 文件，如 pstack，pmap 和 pflags。

（3）使用调试器来获得 core 文件中的堆栈详情。

1. 使用 file

第一步分析 core dump 文件的有用的工具是 file 命令，file 命令尝试用各种文件参数的测试。

有三种测试，执行顺序是：文件系统测试，位数测试，语言测试。

file 命令将打印文件的类型，如果是 core 文件，将得到类似以下的输出。

示例 12-3：

```
core: ELF 32-bit MSB core file SPARC Version 1, from "sleep"
```

这表明是'sleep'可执行文件导致了 core dump。

2. 使用本地堆栈跟踪工具

在不同的平台生成的 core 文件，由不同的本地堆栈跟踪工具来分析，见表 12-1。

表 12-1

平台	pstack	pmap
Solaris	pstack	pmap
AIX 5.2 or higher	procstack	procmap
Linux	lsstack	pmap
HPUX	Not avail	Not avail

如果没有进入调试器，检查操作系统上是否有 pstack 和 pmap 程序，如果有这些程序（在某些操作系统上，用户必须下载这些程序），可以使用这些命令来获取 core 文件的信息。

命令的语法如下所示。

示例 12-4：

```
$ /usr/proc/bin/pstack core
$ /usr/proc/bin/pmap core
```

3. 使用 pstack

pstack 提供了 core 文件的堆栈跟踪信息。

如下为 pstack 输出示例。

示例 12-5：

```
Current function is f get card
1810          tError.nCode = Fvftos32( pFmlBufT, (char *)ptInitCardSend,
"vINIT_CARD_SEND");

(/opt/SUNWspro/bin/../WS6U2/bin/sparcv9/dbx)
(/opt/SUNWspro/bin/../WS6U2/bin/sparcv9/dbx) where

current thread: t@1
[1]  nextfldocc(0x1d1d51,  0x1d0178,  0x1d4060,  0x1d1d68,  0xffbdd364,
0x1d0198), at 0xff1955f4
[2]  copyfbtos(0x1d0190,  0x1d0178,  0x2,  0x1b6588,  0x1c40,  0x1c00),  at
0xff195484
[3] Fvftos32(0x1d0178, 0xffbdde94, 0xcce20, 0x17, 0x91039858, 0x30002), at
0xff194f7c
[4]  f get card(ptEoiCard  =  0xffbecd9f,  ptAddUpdCardRecv  =  0xffbe7998,
ptInitCardSend = 0xffbdde94), line 1810 in "j89.c"
[5] f process card type(pRqst = 0xf00e4, ptEoiCard = 0xffbecd9f), line 636
In "j89.c"
[6] t_eoi_txns(pRqst = 0xf00e4), line 450 in "j89.c"
[7]  tmsvcdsp(0xe44a0,  0xffbed80c,  0x1,  0xc0000000,  0x80000,  0x1),  at
0xff24f8f0
[8]  tmrunserver(0xeebf8,  0xfe7a9430,  0x0,  0x0,  0xeac48,  0xe5ae8),  at
0xff272454
[9]  tmstartserver(0xf, 0xffbed93c, 0xc6bb0, 0x0, 0x0, 0x0), at 0xff24e668
[10] main(0xf, 0xffbed93c, 0xffbed97c, 0xc6800, 0x0, 0x0), at 0x17fc8
```

在 pstack 输出的最上面的行表明导致 core dump 的实际行，然而孤立地分析毫无意义，在这种情况下，最好顺着调用往回跟踪，试着发现问题的根源，在上面的堆栈跟踪中，被 copyfbtos 调用的 nextfldocc API 导致了崩溃，反过来 copyfbtos 被 Fvftos32 调用了。

出错的地方是在 j89.c 文件第 1810 行的 f_get_card 函数中，很可能是调用 Fvftos32 的代码有错误。

名字前面带下划线的函数（如_tmsvcdsp）是 Tuxedo 内部的函数，这些通常不是导致 core dump 的原因。

12.3.4 探查错误的根源

许多调试器需要 Tuxedo 的可执行程序以命令行形式执行，要这样做，需要知道 Tuxedo 传给可执行程序的参数。要得到这些参数，在设置完 Tuxedo 环境变量后，输入以下命令。

示例 12-6：

```
tmboot -i <svrid> -g <group> -n -d1
```

这样将不会启动 SERVER，但这样会显示传递到 Tuxedo 可执行程序的确切的命令行参数。

1. 在 Solaris 上探查

当确认 core 文件是来自应用后，接下来可以运行调试器来获得堆栈信息。Sun 推荐使用 dbx 调试器来获得堆栈跟踪，虽然 GNU’s gdb 也可以提供有用的信息。

使用 dbx 获得堆栈信息示例如下。

示例 12-7：

```
$ dbx exefile corefile [arguments]
```

调试一个给定的可执行程序以及 core 文件，[arguments]指定 Tuxedo 给可执行程序传递的参数。

或者使用如下命令。

示例 12-8：

```
$ dbx exefile processid
```

使用该命令将调试器连接到到正在执行的 Tuxedo 服务进程。

dbx 子命令如下所示。

示例 12-9：

```
(dbx) where (显示堆栈信息)
(dbx) quit  (退出 dbx)
```

2. 在 Linux 上探查

当确认 core 文件是来自应用后，接下来可以运行调试器来获得堆栈信息。Linux 上 GNU’s gdb 是默认的且很好的调试器。

获得堆栈信息的步骤如下。

（1）确保使用最新的 GDB 来避免一些已知的 BUG。

（2）确保 Linux 中 ulimit 设置为 ulimit -c unlimited（core 文件大小无限制）。

（3）在 Linux 中，core dump 默认是关闭的，检查/etc/security 下的文件 limits.conf，打开 core 的设置。

使用 GNU gdb 的示例如下。

示例 12-10：

```
$ gdb  executable  corefile  [arguments]
```

使用 core 文件调试可执行程序，如何使用 arguments 可以参见前面讲述的使用 Solaris dbx 部分。

或者使用如下命令。

示例 12-11：

```
$ gdb exefile processid
```

使用该命令将调试器连接到正在执行的 Tuxedo 服务进程。

gdb 子命令如下所示。

示例 12-12：

```
(gdb) where      (显示堆栈信息)
(gdb) bt         (显示堆栈信息)
(gdb) quit       (退出 gdb)
```

GNU's gdb 很容易使用并且支持很多系统，并可以从 GNU 获得 gdb 手册。

使用 gdb 的 bt 打印程序堆栈例子如下。

示例 12-13：

```
(gdb) bt
#0  AddrSpace::InitRegisters (this=0x51e50) at ../userprog/addrspace.
cc:241
#1  0x00022144 in main (argc=3, argv=0xffbefa34) at ../threads/main.cc:359
(gdb)
```

命令 bt 提供了另一种方法来对 core 文件进行堆栈跟踪。

3. 在 HPUX 上探查

当确认 core 文件是来自应用后，接下来可以运行调试器来获得堆栈信息。HPUX 提供了 adb，当然 GNU's gdb 也将提供有用的信息。

使用 HP-UX adb 示例如下。

示例 12-14：

```
$ adb  executable  corefile  [arguments]
```

使用 core 文件调试可执行程序，如何使用 arguments 见使用 Solaris dbx 部分。

示例 12-15：

```
adb> $c      (显示堆栈信息)
adb> $r      (显示寄存器信息)
adb> $q      (退出 adb)
```

当在 adb 中使用命令$c 时，如果得到一个信息如“can't unwind -- no_entry”，这很有可能是由于 adb 不能识别共享库，出现这种情况时，可以尝试使用 gdb 或者 wdb。

可以从 HP 获得 adb 手册和详细资料。

4. 在 AIX 上探查

当确认 core 文件是来自应用后，接下来可以运行调试器来获得堆栈信息。AIX 上 dbx 可以用来分析 core 文件，GNU's gdb 也可以利用。

其主要基本用法参见 Solaris 上的 dbx，但有些更细节的内容，可以参考 AIX 的官方手册。

5．在 Windows 上探查

在 Windows 系统中，一个 Dr.Watson 日志文件 drwtsn32.log 类似于 UNIX 的 core 文件，该文件生成在如下目录。

示例 12-16：

```
c:\Documents and Settings\All Users\Documents\DrWatson
```

使用 DrWatson 来探查错误原因的步骤如下。

（1）在命令行启动 Dr.Watson：

输入 drwtsn32，Windows 2000 的 Dr. Watson 对话框出现。单击 DrWatson 日志文件预览按钮将解释 drwtsn32.log 的格式。

（2）打开/关闭 Dr.Watson：

默认情况下，Windows NT 安装下 Dr.Watson 是打开的，检查注册表项确保 Dr.Watson 启用（0 表示启用，1 表示未启用）。

示例 12-17：

```
\HKEY_LOCAL_MACHINE\SOFTWARE \Microsoft\Windows NT\CurrentVersion\AeDebug
```

有一个选项“AUTO”可以配置 Dr.Watson 的启动模式，这样它将加载任何调试器，或者应用。

12.4　core dump 成因案例

最常见的 core dump 案例是 Tuxedo 应用程序出现错误导致的。鉴于 Tuxedo 应用都是 C/C++代码，以下 core 例子都是涉及到通常 C 应用开发错误。

12.4.1　为 strings 分配太少的内存

描述：当分配空间给 string 时，不要忘记分配末尾的 null 字节。

一个常见的错误代码如下。

示例 12-18：

```
char *p = (char *) malloc(strlen(str));
strcpy(p, str);
```

示例 12-18 代码没有分配足够的空间给 string，正确的代码如下。

示例 12-19：

```
char *p = (char *) malloc(strlen(str) + 1);
strcpy(p, str);
```

避免：使用 strdup 拷贝 string（但是，这个函数并不是在所有机器上可用，所以这也可能导致问题）。

12.4.2 使用已经释放的内存

描述：当分配的内存用 free()销毁时，这段内存将可能改变，或者被 malloc()重用。在以上任何情况下，如果继续使用已经释放的内存，这段内存将出现异常。

避免：从编程上避免重复释放，或使用内存分析工具排查。

12.4.3 在 scanf 调用丢掉&

描述：当调用 scanf()，参数匹配%d、%u、%o、%x、%i、%n、%e、%g、%f 和%c 必须匹配 integer、float、double 或者 character。比如传递一个 integer 而不是一个指针指向 integer（如 x 而不是&x），将导致 core dump。

避免：使用 lint 检查代码。

12.4.4 用非法的参数调用函数

描述：如果在使用一个函数之前没有声明，C 编译器不知道函数使用哪种参数，或者传递的参数类型不匹配可以导致严重错误。

避免：使用 lint，将指出程序中使用但没有声明的地方，通常使用参数声明函数（例如，不要声明函数如 foo()）。

12.4.5 没有分配内存给指针

描述：声明 char *p，如果没有分配任何内存给 p 而使用 p（例如，strcpy(p, "hi there")），内存将指向一个不可用的地方；

避免：可以使用 lint 检查，将指出变量没有初始化；然而这种方法不会指出 array 元素（或者 structure）。

12.4.6 没有初始化变量

描述：当变量在函数中声明时，函数将其分配在 stack 区，在栈中变量的初始值通常是一个简单的随机值，这个值会在函数调用时改变。

类似的情况将发生在当使用 malloc()或 realloc()分配内存时，在使用前而没有初始化。

避免：可以使用 lint 避免，将指出变量没有初始化，然而这种方法不会指出 array 元素（或者 structure），当分配内存时使用 calloc()代替 malloc()，这将使新分配的内存被赋值为零。

12.5 错误信息的含义

错误信息可能的解释都包含如下。

（1）Bus Error 总线错误。
（2）Memory Fault 内存错误。
（3）IOT Trap I/O 陷阱。
（4）Trace/BPT Trap 跟踪/BPT 陷阱。
（5）Floating Exception 浮点异常。
（6）Segmentation Fault 分段错误。
（7）Illegal Instruction 非法命令。

12.5.1　总线错误

（1）参数的类型或者参数不匹配。
（2）I/O 问题——读 EOF，读取一个关闭的文件、错误的文件指针等。
（3）不当的定位从内存中读。
示例如下。
示例 12-20：

```
int intarray[20];
char a = *(&(intarray[5]) + 1);
//Read the second byte of position 5 of intarray
```

（4）引用不存在的总线设备。
（5）NULL 或者初始化指针，下标超出范围。

12.5.2　内存错误

（1）数组下标超过程序中的内存分配。
（2）NULL 或者初始化指针，下标超出范围。
（3）试图引用有效地址空间以外的数据。
（4）地址空间的奇偶校验误差。
（5）递归（试着惯常的 read()和 write()然后调用 scanf()或者 printf()看看会发生什么）。

12.5.3　I/O 陷阱

执行 I/O 陷阱指令。

12.5.4　跟踪/BPT 陷阱

（1）跟踪位组在处理器的状态（long）字。
（2）试着通知程序访问一个不存在的地址。
（3）肆意破坏导致执行数据或者命令丢失。

12.5.5 浮点异常

（1）非法的浮点运算。
（2）溢出。
（3）下溢。
（4）除以 0。
（5）记录负数。
（6）Float-to-int 转换溢出。

12.5.6 分段错误

（1）下标越界。
（2）处理 printf()，例如，非 null 终止的 string。

12.5.7 非法命令

（1）试着执行越权指令，如 halt()，这些命令只有拥有特殊权限才能执行。
（2）执行无义的命令，如转向累加寄存器。

第 13 章　异常高 CPU 占用率故障

13.1　异常高 CPU 占用率

异常高 CPU 占用率：当一个进程或线程占用的 CPU 资源异常高发生时，可能会使其他进程或线程丧失或缺乏执行要求的处理任务所需的 CPU 处理能力，应进行定期检查。

13.2　异常高 CPU 占用率的伴随症状

（1）用户响应时间长。

（2）Tuxedo 服务器运行速度异常慢。

（3）请求或操作开始出现超时。

13.3　异常高 CPU 占用率探查

13.3.1　探查概述

基本步骤如下。

（1）首先确定哪个进程或线程导致了高 CPU 占用率。

（2）通过各种系统工具或命令探究具体的问题原因。

以下就以最简单的 Tuxedo 程序 Demo simpapp 为例，为了更好地复现 CPU 异常走高，Tuxedo 服务端添加了一个不带 sleep 的死循环。

这时客户端会发现请求始终没有返回，从而导致超时。

示例 13-1：

```
[root@landingbj test]# simpcl test
Can't send request to service TOUPPER
Tperrno = 13
```

此时在服务机输入 top 命令。

示例 13-2：

```
  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM  TIME+    COMMAND
 4843 root      25   0  6424 2512  1712 R  90.0  1.0   1:29.33  simpserv
    1 root      17   0  2072  620  532  S  0.0   0.2   0:01.36  init
```

```
    2 root      RT  -5    0    0    0 S  0.0  0.0   0:00.00 migration/
    3 root      34  19    0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/
    4 root      RT  -5    0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    5 root      10  -5    0    0    0 S  0.0  0.0   0:00.01 events/0
    6 root      12  -5    0    0    0 S  0.0  0.0   0:00.00 khelper
    7 root      13  -5    0    0    0 S  0.0  0.0   0:00.00 kthread
   10 root      10  -5    0    0    0 S  0.0  0.0   0:00.02 kblockd/0
```

很明显，进程 6991（即 simpserv 所属进程）导致 CPU 占用率很高。

为了探究具体是哪一个系统调用或 API 上引起的问题，下面在几种不同操作系统上分别进行探查。

13.3.2 初步探查

1. 在 Linux 上初步探查

可以运行 strace 命令探查系统调用。

示例 13-3：

```
strace -o outfile -p <PID>
```

例如，针对上述 top 结果中在第一位的 simpserv 如下。

示例 13-4：

```
strace -o strace.out -p 4843
```

2. 在 HP-UX 上初步探查

可以运行 tusc 命令探查进程的系统调用，使用 tusc 可以获得进程系统调用的所有信息，HP-UX 也提供了 truss 和 tusc 一起打包，命令行如下。

示例 13-5：

```
truss -d -o <outfile> -p <pid>
```

“-d”参数是列出所有系统调用的时间戳。

3. 在 Solaris/AIX 上初步探查

Solaris 或 AIX 上也可以使用 truss 命令查看系统调用。

13.3.3 进一步跟踪

操作系统提供了很多工具可以辅助用户进行追踪，如 truss、strace、gdb、dbx 等系统调用工具。

所有的 UNIX 程序都是通过调用操作系统提供的这些底层服务来完成它们的任务，使用这些工具，可以清楚地看到这些调用过程及其使用的参数。

通过这种方式，也可以了解程序与操作系统之间的底层交互。

示例 13-6：

```
strace -p 4843
Process 4843 attached - interrupt to quit
msgrcv(65538, 65538, {536870962, "p\0\0\0\2\0\1\0\0\0\0\0\270\1\0\0\0\
0\0\200\0\0\0\0\0\0\0\0X\230\3\221"...}, 4572, -1073741824, 0) = 436
 llseek(8, 8704, [8704], SEEK SET)      = 0
read(8,  "\375\365\0\0\310\0\0\0,\1\0\0\0\0\0\0\244\0\0\0\363.\0\01\1\0\
0\330\1\0\0"..., 4096) = 4096
futex(0x293440, FUTEX WAKE PRIVATE, 2147483647) = 0
futex(0x441844, FUTEX WAKE PRIVATE, 2147483647) = 0
 llseek(8, 4608, [4608], SEEK SET)      = 0
read(8, "\270\26\0\0\310\0\0\0004\1\0\0\0\0\0\0\244\0\0\0}\351\0\01\1\0\
0000\3\0\0"..., 4096) = 4096
futex(0x441ac0, FUTEX WAKE PRIVATE, 2147483647) = 0
futex(0x4418a4, FUTEX WAKE PRIVATE, 2147483647) = 0
time(NULL)                             = 1302007088
access("/home/Tuxedo/test/ULOG.040511", F OK) = 0
open("/home/Tuxedo/test/ULOG.040511", O WRONLY|O CREAT|O APPEND|O
LARGEFILE, 0666) = 3
write(3, "053808.landingbj!simpserv.4843."..., 131) = 131
close(3)                               = 0
Process 4843 detached（此处为 Ctrl+C 结束）
```

当执行到 simpserv 时一直阻塞在此处，不往下执行任何操作（表现：不再往下打印各种系统调用信息）。

再比如，gdb 等程序调试命令（确定问题后，用于调试问题程序）。

示例 13-7：

```
gdb --quiet
(gdb) attach 4843
Reading symbols from /home/Tuxedo/test/simpserv...(no debugging symbols
found)...done.
Reading symbols from /home/Tuxedo/tux11/tuxedo11gR1/lib/libtux.so...done.
Loaded symbols for /home/Tuxedo/tux11/tuxedo11gR1/lib/libtux.so
Reading  symbols  from  /home/Tuxedo/tux11/tuxedo11gR1/lib/libbuft.so...
done.
Loaded symbols for /home/Tuxedo/tux11/tuxedo11gR1/lib/libbuft.so
Reading symbols from /home/Tuxedo/tux11/tuxedo11gR1/lib/libfml.so...done.
Loaded symbols for /home/Tuxedo/tux11/tuxedo11gR1/lib/libfml.so
Reading  symbols  from  /home/Tuxedo/tux11/tuxedo11gR1/lib/libfml32.so...
done.
Loaded symbols for /home/Tuxedo/tux11/tuxedo11gR1/lib/libfml32.so
Reading   symbols   from   /home/Tuxedo/tux11/tuxedo11gR1/lib/libengine.so
...done.
Loaded symbols for /home/Tuxedo/tux11/tuxedo11gR1/lib/libengine.so
Reading symbols from /lib/libdl.so.2...done.
Loaded symbols for /lib/libdl.so.2
Reading symbols from /lib/libpthread.so.0...done.
```

```
[Thread debugging using libthread db enabled]
[New Thread 0xb7f9d8d0 (LWP 4843)]
Loaded symbols for /lib/libpthread.so.0
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /home/Tuxedo/tux11/tuxedo11gR1/lib/libutrace.so
...done.
Loaded symbols for /home/Tuxedo/tux11/tuxedo11gR1/lib/libutrace.so
Reading symbols from /home/Tuxedo/tux11/tuxedo11gR1/lib/libgiconv.so
...done.
Loaded symbols for /home/Tuxedo/tux11/tuxedo11gR1/lib/libgiconv.so
Reading symbols from /home/Tuxedo/tux11/tuxedo11gR1/lib/libusort.so
...done.
Loaded symbols for /home/Tuxedo/tux11/tuxedo11gR1/lib/libusort.so
Reading symbols from /lib/libnsl.so.1...done.
Loaded symbols for /lib/libnsl.so.1
Reading symbols from /lib/ld-Linux.so.2...done.
Loaded symbols for /lib/ld-Linux.so.2
Reading symbols from /home/Tuxedo/tux11/tuxedo11gR1/lib/registry.so
...done.
Loaded symbols for /home/Tuxedo/tux11/tuxedo11gR1/lib/registry.so
0x08048710 in TOUPPER ()
```

attach 4843 命令启动对当前正在运行的 Tuxedo 服务器的调试工作。

用 info 命令则列出程序信息如下。

示例 13-8：

```
info proc
process 4843
cmdline = 'simpserv'
cwd = '/home/Tuxedo/test'
exe = '/home/Tuxedo/test/simpserv'
```

info functions 命令得到函数列表如下。

示例 13-9：

```
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x080484ac   init
0x080484d4    gmon start   @plt
0x080484e4  toupper@plt
0x080484f4    libc start main@plt
0x08048504  tpsvrdone
0x08048504  tpsvrdone@plt
```

```
0x08048514   tmstartserver@plt
0x08048524   tmrunserver
0x08048524   tmrunserver@plt
0x08048534  userlog@plt
0x08048544  tpsvrthrdone
---Type <return> to continue, or q <return> to quit---Type
0x08048544  tpsvrthrdone@plt
0x08048554  tpsvrthrinit
0x08048554  tpsvrthrinit@plt
0x08048570   start
0x08048594  call_gmon_start
0x080485c0  __do_global_dtors_aux
0x08048620  frame dummy
0x08048644   tmgetsvrargs
0x0804866c  main
0x080486a4  tpsvrinit
0x080486bd  TOUPPER
```

以上是 simpserv 应用程序部分信息。

通过 where 命令获取进程堆栈信息。

示例 13-10：

```
(gdb) where
#0  0x08048710 in TOUPPER ()
#1  0x00171bf5 in  tmsvcdsp ()
    from /home/Tuxedo/tux11/tuxedo11gR1/lib/libtux.so
#2  0x0019c22c in  tmrunserver ()
    from /home/Tuxedo/tux11/tuxedo11gR1/lib/libtux.so
#3  0x0016f72d in  tmstartserver ()
    from /home/Tuxedo/tux11/tuxedo11gR1/lib/libtux.so
#4  0x0804869a in main ()
(gdb) detach
Detaching from program: /home/Tuxedo/test/simpserv, process 4843
(gdb) quit
```

13.4　异常高 CPU 占用率故障排除策略

（1）通过重复收集信息确定 CPU 占用率异常。

（2）在服务器端通过系统命令查找 CPU 占用率过高的进程。

（3）通过各种系统工具对进程的分析找出导致高 CPU 占用率的具体原因。

（4）继续监视，看是否还会发生故障。

（5）控制客户端超时时间（UBBCONFIG 中 WSL CLOPT [-t timeout]），以减少系统故障的危害。

第 14 章　常规服务器阻塞故障

14.1　确认是服务器阻塞

Tuxedo 应用可能处于挂起状态，从而表现为如下情况。

（1）不能完成现在执行的任务。

（2）不能响应新的请求。

（3）只占有很少的 CPU 时间。

14.2　服务器阻塞的可能原因

一般来讲，Tuxedo 的服务进程 Server 挂起是因为等待某些资源。当缺少某种资源或者该资源上有互斥锁，进程不能获得资源，就会挂起，从而不能响应新的请求。

例如，一个进程想使用共享内存，它将一直等到其他进程释放锁的信号量，不然等待的进程将挂起。

服务器阻塞的可能原因如下。

（1）死锁。

（2）阻塞在资源上。

（3）睡眠循环。

（4）超时。

14.3　服务器阻塞的探查

问题探查的目的是搜索挂起进程的信息，分析信息找出挂起的原因。

怎么识别一个进程是否挂起？问题的探查步骤如下。

（1）运用 Tuxedo 管理工具 tmadmin。

① 使用 pq 检查 Tuxedo 队列中的请求数。

② 使用 psr 检查 Tuxedo Server 的状态。

（2）检查 Tuxedo SERVER 的 CPU 使用状况，使用操作系统工具。

① 使用操作系统调试工具找出挂起 SERVER 的详细系统调用或者 API，如 truss、strace、gdb、dbx 等。

② 或者，发送一个 kill 信号 SIGABRT 来生成一个 core dump，这样可以使用调试工

具来调试 core 文件以发现进程挂在哪里。

一般的原则性方法是相通的，但对于每个具体不同的操作系统，有些细微的差别，以下是针对不同操作系统的具体步骤。

14.3.1　Solaris

1．运行 tmadmin 检查 SERVER 状态

（1）使用 pq 收集请求队列中的请求信息

例如，echo pq | tmadmin

可以使用以下脚本辅助数据收集 pq.sh [t] [n]（输出结果间隔 t 秒输出 n 次）。

示例 14-1：

```
#!/usr/bin/sh
if test -z "$1"
then
sleep_time=0
else
    sleep time=$1
fi
if test -z "$2"
then
    loopnum=1
else
    loopnum=$2
fi
num=0
while [ $num -lt $loopnum ]
do
    num='echo "$num + 1" | bc'
    echo pq | tmadmin 2>/dev/null
    sleep $sleep_time
done
```

示例 14-2：

```
输出：
$ pq.sh 5 3
> Prog Name  Queue Name  # Serve  Wk Queued  # Queued  Ave. Len  Machine
---------    -------------------  ---------  --------  -------   -------
simpserv     00001.00100          1      100         2       0.0    simple
BBL          222222               1        0         0       0.0    simple

> Prog Name  Queue Name  # Serve  Wk Queued  # Queued  Ave. Len  Machine
---------    -------------------  ---------  --------  --------  -------
```

```
simpserv      00001.00100      1        250        5        0.0     simple
BBL           222222           1          0        0        0.0      simple

> Prog Name   Queue Name   # Serve   Wk Queued   # Queued   Ave. Len   Machine
---------     -------------------   ---------   ---------  ---------  -------
simpserv      00001.00100      1        400        8        0.0     simple
BBL           222222           1          0        0        0.0      simple
```

可以看到 simpserv 的#Queued 列的值一段时间内一直在增长而没有下降，可能有阻塞问题。

simpserv 的队列名是：00001.00100。

（2）使用 psr 找出挂起服务的名字

示例 14-3：

```
echo psr -q <queue_name> | tmadmin 2>/dev/null | grep <process_name>
$ echo psr -q 00001.00100 | tmadmin 2>/dev/null | grep simpserv
simpserv   00001.00100   GROUP1   100   0   0   TOUPPER
```

该程序表示 SERVER 在运行客户端的请求。

运行的是 TOUPPER 服务，SRV_ID 是 100。

（3）使用 tmadmin 探查 SVR_ID

示例 14-4：

```
$ tmadmin
tmadmin - Copyright (c) 1996-1999 BEA Systems, Inc.
Portions * Copyright 1986-1997 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
Tuxedo is a registered trademark.

> verbose
Verbose now on.

> psr -i 100
Group ID: GROUP1, Server ID: 100
Machine ID: simple
Process ID: 16979, Request Qaddr: 905, Reply Qaddr: 905
Server Type: USER
Prog Name: /home/landingbj/apps/simpapp/simpserv
Queue Name: 00001.00100
Options: ( none )
Generation: 1, Max message type: 1073741824
Creation time: Fri Sep 24 00:44:42 2004
Up time: 0:06:26
Requests done: 0
```

```
    Load done: 0
    Current Service: TOUPPER
```

这样就可以得到问题中要得到的 Tuxedo SERVER 进程 ID，该例子中的 PID 是 16979。

2．运行 ps 命令确认 Tuxedo SERVER 的 PID

示例 14-5：

```
    $ ps -ef | grep simpserv
    yhuang 16979 1 0 22:56:28 pts/10 0:00 simpserv -C dom=simpapp -g 1 -i 1 -u
slsol3 -U /home/landingbj/apps/simpapp/ULOG -
```

3．运行 prstat（Solaris 的系统工具）探查该进程的 CPU 使用率

示例 14-6：

```
    $ prstat -L -p 16979 1 1
    PID     USERNAME  SIZE    RSS     STATE   PRI NICE TIME     CPU PROCESS/LWPID
    16134  landingbj  6256K  3520K   sleep   59   0   0:00.00  0.0% simpserv/5
    16134  landingbj  6256K  3520K   sleep   58   0   0:00.00  0.0% simpserv/4
    16134  landingbj  6256K  3520K   sleep   45   0   0:00.00  0.0% simpserv/3
    16134  landingbj  6256K  3520K   sleep   56   0   0:00.00  0.0% simpserv/2
    16134  landingbj  6256K  3520K   sleep   58   0   0:00.00  0.0% simpserv/1
```

4．运行 pstack 命令查看堆栈信息

pstack 命令能够列出特定进程的堆栈信息。main 函数将在运行系统调用堆栈的开始，堆栈将帮助识别是哪个系统调用或者 API 导致了 Tuxedo SERVER 挂起。

例如，在下面的堆栈信息中，可以找出 SERVER 挂起在系统调用“sleep”。

示例 14-7：

```
    $ pstack 16979
    16979: simpserv -C dom=simpapp -g 1 -i 1 -u slsol3 -U /home/landingbj/apps/simpa
    ----------------- lwp# 1 / thread# 1 --------------------
    fef1f004 lwp sema wait (20fe0)
    fee39ac4  park (20fe0, fee5e000, 0, 20f20, 24d84, 0) + 114
    fee3978c _swtch (20f20, 0, fee5e000, 5, 1000, 0) + 424
    fee37e10 cond_reltimedwait (0, 0, 0, 1, 0, 0) + 1f8
    fee496c4 sleep (0, fe28c6e8, 44340, ff3e7fe8, fee5e000, fef273d0) + 17c
    00010a78 TOUPPER (2d20c, ffbef7ec, ffbef7ee, 3, 0, 5) + 68
    ff24f8f0  tmsvcdsp (215c8, ffbef8d4, 0, c0000000, 80000, 1) + e58
    ff272454  tmrunserver (2bd20, ff129430, 0, 0, 27d70, 22c10) + 1064
    ff24e668  tmstartserver (e, ffbefa04, 20ce8, fee9bbd0, 31ea0, 0) + 1b0
    00010990 main (e, ffbefa04, ffbefa40, 20c00, 0, 0) + 20
    000108f8 _start (0, 0, 0, 0, 0, 0) + 108
```

14.3.2 Linux

1. 运行 tmadmin 检查 SERVER 状态

使用和 solaris 上同样的步骤。

2. 运行 ps 命令检查进程状态

ps -e -o pid,user,sz,pcpu,state,args | grep <process_name> or <PID>
示例 14-8：

```
$ ps -e -o pid,user,sz,pcpu,state,args | grep simpserv
PID   USER        SZ   %CPU  S COMMAND
17553 landingbj  1098  0.0   S simpserv -C dom=site1 -g 2 -i 100 -u dell40 -U /usr/
```

%CPU：进程使用系统 CPU 的百分比。
S　　：进程状态。
D　　：不可中断的闲置状态（通常是 IO）。
R　　：运行态（在运行队列中）。
S　　：睡眠态。
T　　：跟踪态或停止状态。
Z　　：死亡的进程（僵尸进程）。

3. 运行 top 命令列出进程 CPU 使用率

top -p <PID> -n 20
示例 14-9：

```
$ top -p 17553 -n 10
PID    USER        PRI NI SIZE RSS   SHARE STAT %CPU %MEM TIME COMMAND
17553  landingbj 15   0  2116 2116  1504  S    0.0  0.0  0:00 simpserv
```

4. 运行 gdb 命令来获得进程堆栈信息

gdb <prog path> <PID>
prog path 是执行文件的路径和名字
示例 14-10：

```
$ gdb simpserv 17553
(gdb) where
#0 0x402b8cb1 in nanosleep () from /lib/libc.so.6
#1 0x402b8b31 in sleep () from /lib/libc.so.6
#2 0x08048971 in TOUPPER (rqst=0x0) at simpserv.c:41
#3 0x40074775 in _tmrunserver () from /usr/tuxedo/tuxedo8.0/lib/libtux.so
```

```
#4 0x400574f5 in  tmstartserver () from /usr/tuxedo/tuxedo8.0/lib/libtux.so
#5 0x0804892a in main ()
#6 0x40219727 in   libc start main () from /lib/libc.so.6
(gdb) detach
(gdb) quit
```

也可以运行 strace 命令探查系统调用。

例如：strace -o outfile -p <PID>

示例 14-11：

```
$ strace -o strace.out -p 17553
$ cat strace.out
rt_sigprocmask(SIG_BLOCK, [CHLD], [RTMIN], 8) = 0
rt_sigaction(SIGCHLD, NULL, {SIG_DFL}, 8) = 0
rt sigprocmask(SIG SETMASK, [RTMIN], NULL, 8) = 0
nanosleep({1000, 0}, <unfinished ...>
```

14.3.3　AIX

1．运行 tmadmin 检查 SERVER 状态

使用和 solaris 上同样的步骤。

2．使用 ps 命令获得进程的 CPU 使用率

ps aux | grep <process name>

示例 14-12：

```
$ ps aux | head -n 1; ps aux | grep simpserv
USER        PID %CPU %MEM SZ  RSS  TTY    STAT STIME    TIME   COMMAND
landingbj  3908 0.0  0.0  904 1036 pts/2 A    17:15:26 0:00   simpserv -C dom=A
```

3．运行 dbx 命令来获得进程堆栈信息

运行 dbx 命令来查看挂起的进程，进入 dbx 后输入“where”，这将输出堆栈信息。在退出 dbx 之前输入“detach”，这样就断开与进程的连接，再退出 dbx。

dbx -a <PID>

示例 14-13：

```
$ dbx -a 3908
stopped in  p nsleep at 0xd0013b34 ($t1)
0xd0013b34 ( p nsleep+0x10) 80410014 lwz r2,0x14(r1)
(dbx) where
_p_nsleep(??, ??) at 0xd0013b34
raise.nsleep(??, ??) at 0xd018560c
```

```
sleep(??) at 0xd01e0250
TOUPPER(0x2002a38c), line 45 in "simpserv.c"
_tmsvcdsp() at 0xd3741b48
_tmrunserver() at 0xd36f30c4
_tmstartserver() at 0xd37a5e94
main(0x12, 0x2ff227bc) at 0x100003f0
(dbx)deatch
```

14.3.4 HP-UX

1. 运行 tmadmin 检查 SERVER 状态

使用和 solaris 上同样的步骤。

2. 运行 ps 命令获得运行进程的 PID

ps -ef | grep <process_name>
示例 14-14：

```
$ ps -ef | grep simpserv
landingbj   17054   1   0   15:31:24   ?   0:00   simpserv -C dom=tux ora -g
2 -i 100 -u bea-cs -U /home landingbj
```

输出的第二列是 PID，值是 17054。

3. 运行 ps 命令检查 SERVER 进程的状态

设置环境变量：export UNIX95=XPG4。

ps -e -o pid,user,sz,pcpu,state,args | grep <process name | PID>
示例 14-15：

```
$ ps -e -o pid,user,sz,pcpu,state,args | grep 17054
PID   USER       SZ  %CPU  S  COMMAND
17054 landingbj  73 0.02 S simpserv -C dom=tux ora -g 2 -i 100 -u bea-cs -U /home
landingbj
```

%CPU：进程占用系统 CPU 的百分比。
S　：进程状态。
0　：不存在。
S　：睡眠态。
W　：等待。
R　：运行。
I　：中间态。
Z　：中止。
T　：停止。

4．运行 tusc 命令探查进程的系统调用

HP-UX 的 tusc 工具可以从以下 URL 下载。

http://www.hp.com/workstations/segments/mcad/dassault/plmcc/perf_tools.html

使用 tusc 可以获得进程系统调用的所有信息，HP UNIX 也提供了 truss 和 tusc 一起打包，命令行如下：

```
truss -d -o <outfile> -p <pid>
```

-d 参数是列出所有系统调用的时间戳。

当检查 truss 输出时发现：SERVER 进程阻塞在 sigtimedwait API，迟一秒后，该系统调用就被一个信号打断，现在时间就被系统函数 time 获得，然后 sigtimedwait 函数调用又到下一个循环，这样就知道进程挂起在一个 sleep 函数的循环。

示例 14-16：

```
$ truss -o 17054 .out -p 17054
Attached to process 17054 ("simpserv -C dom=tux_ora -g 2 -i 100 -u bea-cs
-U /home/ landingbj /") [32-bit] )
0.0000 sigtimedwait(0x7b040ef0, NULL, 0x7b040f10) [sleeping]
0.8234 sigtimedwait(0x7b040ef0, NULL, 0x7b040f10) ERR#11 EAGAIN
0.8239 time(NULL) = 1032847337
0.8240 time(NULL) = 1032847337
1.8334 sigtimedwait(0x7b040ef0, NULL, 0x7b040f10) ERR#11 EAGAIN
1.8336 time(NULL) = 1032847338
1.8337 time(NULL) = 1032847338
2.8435 sigtimedwait(0x7b040ef0, NULL, 0x7b040f10) ERR#11 EAGAIN
2.8490 time(NULL) = 1032847339
2.8680 time(NULL) = 1032847339
3.8734 sigtimedwait(0x7b040ef0, NULL, 0x7b040f10) ERR#11 EAGAIN
3.8736 time(NULL) = 1032847340
3.8738 time(NULL) = 1032847340
4.8906 sigtimedwait(0x7b040ef0, NULL, 0x7b040f10) ERR#11 EAGAIN
4.8908 time(NULL) = 1032847341
4.8910 time(NULL) = 1032847341
```

14.3.5　Windows

1．运行 ipcs 命令检查队列中的信息

ipcs -qob

第 7 列输出（QNUM），列出消息个数。

示例 14-17：

```
D:\Projects\testcase\simpapp>ipcs -qob
```

```
IPCS status from BEA segV8.1 as of Sat Sep 25 01:18:18 2004
T  ID      KEY          MODE        OWNER  GROUP  CBYTES  QNUM   QBYTES
Message Queues:
q  2305  0x0001e242  -Rrw-rw-rw- 0      0      0       0     65536
q  3074  0x00000000  --rw-rw-rw- 0      0      292     1     65536
q  3843  0x00000000  -Rrw-rw-rw- 0      0      292     1     65536
q  5636  0x00000000  --rw-rw-rw- 0      0      292     1     65536
q  2309  0x00000000  -Rrw-rw-rw- 0      0      292     1     65536
```

找出 QNUM 不断增长的 Message Queue ID，如 3843。

2．运行 tmadmin 命令查找该 message queue ID 的 SERVER

tmadmin < psr.txt

psr.txt 将包含两行，如

verbose

psr

当 verbose 打开，psr 命令将列出 Tuxedo SERVER 的详细信息包括进程 ID。

示例 14-18：

```
D:\Projects\testcase\simpapp> tmadmin < psr.txt | findstr 3843
Process ID: 2008, Request Qaddr: 3074, Reply Qaddr: 3843
```

3．使用 pslist 工具

获得挂起进程占用的 CPU 时间，pslist 工具：

http://www.sysinternals.com/ntw2k/freeware/pslist.shtml

pslist <PID|process name>

示例 14-19：

```
>pslist 2008
Name       Pid    Pri   Thd   Hnd    Priv   CPU Time       Elapsed Time
simpserv   2008   8     1     128    780    0:00:00.040    0:38:55.348
```

4．使用 strace 命令

探查进程的堆栈信息，strace 命令：

http://www.bindview.com/Support/RAZOR/Utilities/Windows/strace_readme.cfm

strace -p <PID>

示例 14-20：

```
>strace -p 2008
1   356   324   NtDelayExecution (0, {-100000000, -1}, ... ) == 0x0
2   356   324   NtDelayExecution (0, {-100000000, -1}, ... ) == 0x0
3   356   324   NtDelayExecution (0, {-100000000, -1}, ... ) == 0x0
```

```
    4   356   324   NtDelayExecution (0, {-100000000, -1}, ... ) == 0x0
    5   356   324   NtDelayExecution (0, {-100000000, -1}, ... ) == 0x0
```

14.4　故障排查清单

（1）当 SERVER 挂起时分析进程堆栈信息。
（2）分析以及调试源程序。
（3）检查是否缺少资源。
（4）增加更多的调试代码到源程序中。
（5）检查操作系统的补丁是否正确。

14.5　进程挂起例子分析

14.5.1　进程挂起在 sleep 循环中

运行 truss 或者 strace 来探查问题，可以发现 sleep 系统函数调用另外一个系统调用，该系统调用就阻塞了。（HP-UX 上系统调用可能是 sigtimedwait()）

当睡眠到时，操作系统将发送一个信号到进程，然后这个系统调用将被打断，并且返回一个错误，错误号为 EAGAIN。

如果运行 gdb 或者 dbx 探查进程，将在 where 生成的堆栈信息中发现 sleep 命令被调用。

14.5.2　进程一直等待数据库查询大数据

如果数据库和 Tuxedo 部署在同一个节点上，它们之间通过 IPC 资源交换数据，因此可以发现进程阻塞在和 IPC 相关的系统调用。

如果 Tuxedo 接入数据库是通过 socket，因此发送 SQL 请求的函数将通过系统调用 read 和 write 返回。

14.5.3　死锁：不同 SERVER 中的服务相互调用

1．场景

假设有两 Tuxedo SERVER 名为 SVR_A 和 SVR_B，两个都有很多 SERVICE，例如 SVR_A 有 SVCA1 和 SVCA2，SVR_B 有 SVCB1 和 SVCB2。

如果 Tuxedo 应用中有这样的逻辑：

SVCA1--tpcall()--> SVCB1

SVCB2- -tpcall()- -> SVCA2

当只启动单一 SERVER，即一个 SVR_A 和一个 SVR_B，而客户端应用同时调用 SVCA1 和 SVCB2，这样 SVR_A 和 SVR_B 都可能被挂起。

2. 分析

当客户端 tpcall()一个 SERVER，请求信息将发送到该 SERVER 的请求队列，然后这个进程调用 msgrecv()将该信息调出队列并且处理该请求，在进程调用 tpreturn()之前，在请求队列没有监听进程。

然而在这个场景中，客户端同时调用 SVR_A 和 SVR_B，这时 SVCA1 tpcall() SVCB1，但同时 SVR_B 正准备处理信息需要调用 SVR_A，因为 SVCB2 tpcall() SVCA2，这样 SVR_A 发送信息给 SVR_B 并且保持阻塞状态一直等到获得 SVR_B 的返回结果，在这同时，SVR_B 发送信息给 SVR_A 并且保持阻塞状态一直等到获得 SVR_A 的返回结果。启动的 SVR_A 和 SVR_B 将都处于阻塞状态。此时 SVR_B 不能处理 SVR_A 的请求，SVR_A 不能处理 SVR_B 的请求，这就是死锁状态。

3. 解决方案

创建 SVCA2 和 SVCB1 SERVICE 到两个新的 SERVER 中，确保两个 SERVER 之间没有依赖。

第 15 章　内存不足和内存泄漏故障

15.1　问题描述

内存泄漏通常是指计算机程序无法释放不再使用的内存。严格说来，这种行为是内存消耗过度。内存泄漏发生在程序失去内存消耗平衡的能力。这种行为通常会降低计算机的性能，因为它无法使用所有可用内存。

内存泄漏是程序的一个常见错误，尤其是发生在没有垃圾回收机制的编程语言，如 C 和 C++，它们很大程度依赖于指针操作。

主要的问题在于通常操作系统的组件在负责管理内存，因此内存泄漏的结果是系统会从总体上使用更多的内存，而不仅仅只是这个进程或程序。这会导致消耗过多的系统可用内存或者使重要的子系统停止运转。

一些语言提供自动垃圾收集机制，如 Java、C#和 LISP。但这些语言同样可能导致内存泄漏。内存管理程序无法释放一个仍然被其他对象引用的对象（例如，把一个对象存储在 Java Vector 中，然后丢失/忘记索引）。开发者应该负责清理使用后的对象。然而，对于开发者而言，自动内存管理更加方便，由于应用程序需要做额外的检查工作，所以垃圾收集会影响应用程序的性能。

15.2　问题诊断

以下会引入导致内存泄漏的代码，以及进入如何找出解决方法的途径的概述。

15.2.1　进程地址空间及物理内存的区别

每个进程都有其自身的地址空间。在 32 位的操作系统中，这个空间范围在 0 到 4GB 之间。机器的可用 RAM 或 swap space 的地址空间是相互独立的。机器的总的可用物理内存是 RAM 与 swap space 的总和。所有运行中的进程共享物理内存。

进程中的地址空间是虚拟的。操作系统内核把虚拟地址映射到物理地址。物理地址值指向物理内存中的某一个位置。在任何时候，机器上运行的进程虚拟内存的总和不超过机器上总的可用的物理内存。

15.2.2　为什么这个问题会发生

很大程度上，内存泄漏都是程序的错误。当程序运行时，存在微小的 bug 并不是太显

著。根据泄漏的大小，在 Tuxedo 服务器从开始运行，到出现内存泄漏的表现时，可能需要花费数小时甚至数天的时间。当应用程序开发者分配内存，但没有回收内存时问题就会产生。

看一下如下的 C 代码。

示例 15-1：

```
int main(void)
{
   char *string1 = malloc(sizeof(char)*50);  // a)
   char *string2 = malloc(sizeof(char)*50);  // b)
   scanf("%s", string2);
   string1 = string2; // c)

   free(string2); //d)
   free(string1); //e)
   return 0;
 }
```

以上例子中分配给 string1 的内存会丢失。

（1）内存分配给 string1，返回内存地址给 string1 变量（把它称之为 address1，因此 string1->address1）;

（2）内存分配给 string2，返回内存地址给 string2 变量（把它称之为 address2， 因此 string2->address2）;

（3）这里把 sting2 变量赋值给 string1。这个操作以后 string1 和 string2 指向同一个地址（string1->address2, string2->address2）。在这点上 address1 内存发生泄漏，因为无法引用并且释放它。

（4）这个释放内存的操作会成功，因为 string2 指向 address2，它有被分配内存。在这次调用之后 address2 返回给操作系统。

（5）这个调用会失败，string1 同样指向 address2。而在 address2 的内存已经被（4）过程释放，故调用会失败。

从以上例子可以看出，给 string1 分配的内存无法被释放，因为没有变量来引用这块内存。

15.3 问题研究

内存泄漏并不是太好诊断。主要的方法为使用如 insure++、Purify 或者 Valgrind 等内存泄漏分析工具。

内存泄漏的症状包括如下方面（但是并不限于这几个方面）。

（1）响应时间逐步恶化，直到导致程序失败。

（2）日志信息表明进程已经停止或者没有响应。

（3）错误信息表明系统的虚拟内存很低。在 Windows 机器上，错误信息通常是一个弹窗，如图 15-1 所示。



图 15-1

当应用程序运行时，如果发现一些或者所有的这些症状的出现，那么就有可能是内存泄漏了。

15.4 分析与检测内存泄漏

以下 3 个步骤用来检测与分析内存泄漏。

（1）监控程序进程的虚拟内存大小，如果进程的虚拟内存的值在一段时间内持续增长，那么就可能是内存泄漏。

（2）隔离应用程序的各个组件尝试找出内存泄漏的位置。

（3）如果步骤（2）无法找出内存泄漏的具体位置，那么使用内存监控工具找出泄漏。

下面的几节将更详细地讨论以上几个主要步骤。

15.4.1 监控进程虚拟内存大小

每个启动状态的 Tuxedo 服务器会产生一个进程，不管是在 Windows 还是在 UNIX 平台下。用户可以监控这些进程的虚拟内存使用情况来判断是否持续增长。如下是平台相关的一些方法来监控进程虚拟内存的使用情况。

1. Windows

（1）Windows 任务管理器

一种找出一个进程虚拟内存使用的简单方法是使用任务管理器。可以使用以下几种方法打开它。单击“开始”按钮，选择“运行”命令，在运行对话框中输入 taskmgr。或者右键单击 Windows 任务栏，选择“任务管理器”命令。任务管理器窗口与图 15-2 类似。

在进程标签中列出了当前系统中运行的所有进程，进程虚拟内存的使用情况通过“查看”->“选择列”，在复选框中选上“虚拟内存”一栏，然后单击确定。进程列表就会显示系统中每个进程的虚拟内存大小。通过监控分配的虚拟内存大小可以看到一个进程是否随着时间的推移消耗更多的内存。使用 Windows 任务管理器时还有其他许多关于内存的选项如分页默认值、分页默认差值、内存利用率、内存利用率差值，它们也可以作为诊断内存问题的参考信息。

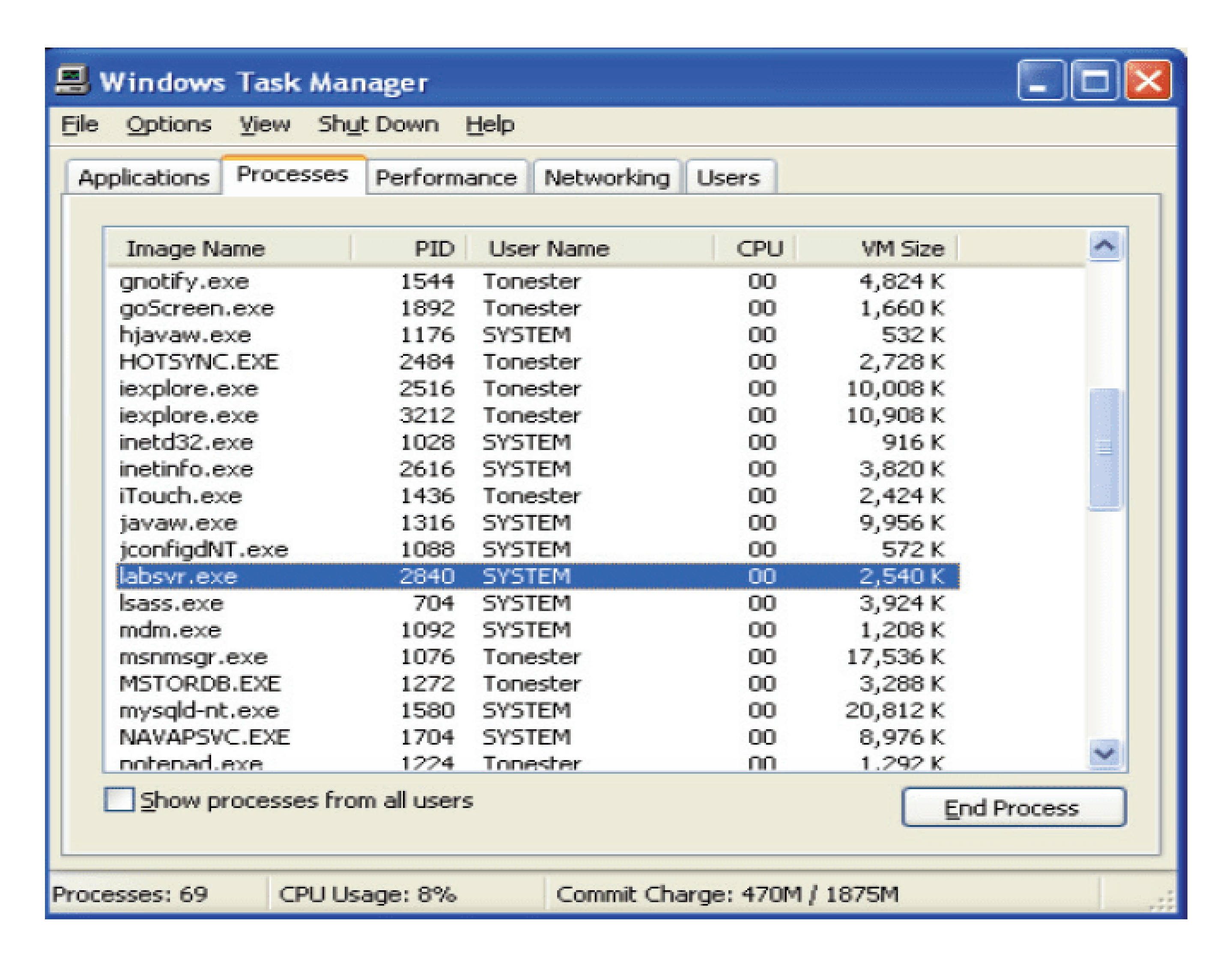


图 15-2

（2）Windows 性能监视器

性能监视器是微软管理控制台的一部分。这个控制台可以用于监控系统中进程的内存利用率。要使用性能监视器监视一个进程内存的利用率，执行以下步骤。

① 在开始菜单中选择“运行”命令，在对话框中输入“perfmon”，单击“确定”按钮。

② 在弹出的性能窗口中单击“+”号按钮（在图表的上方）。

③ 在对话框中选择选项为：性能对象：process（不是默认的 processor）。

④ 选择计数器：Virtual Bytes。

⑤ 从列表中选择实例：选择选择 Tuxedo 服务器的实例。

⑥ 单击“增加”按钮，然后单击“关闭”按钮。

性能监控工具如图 15-3 所示。

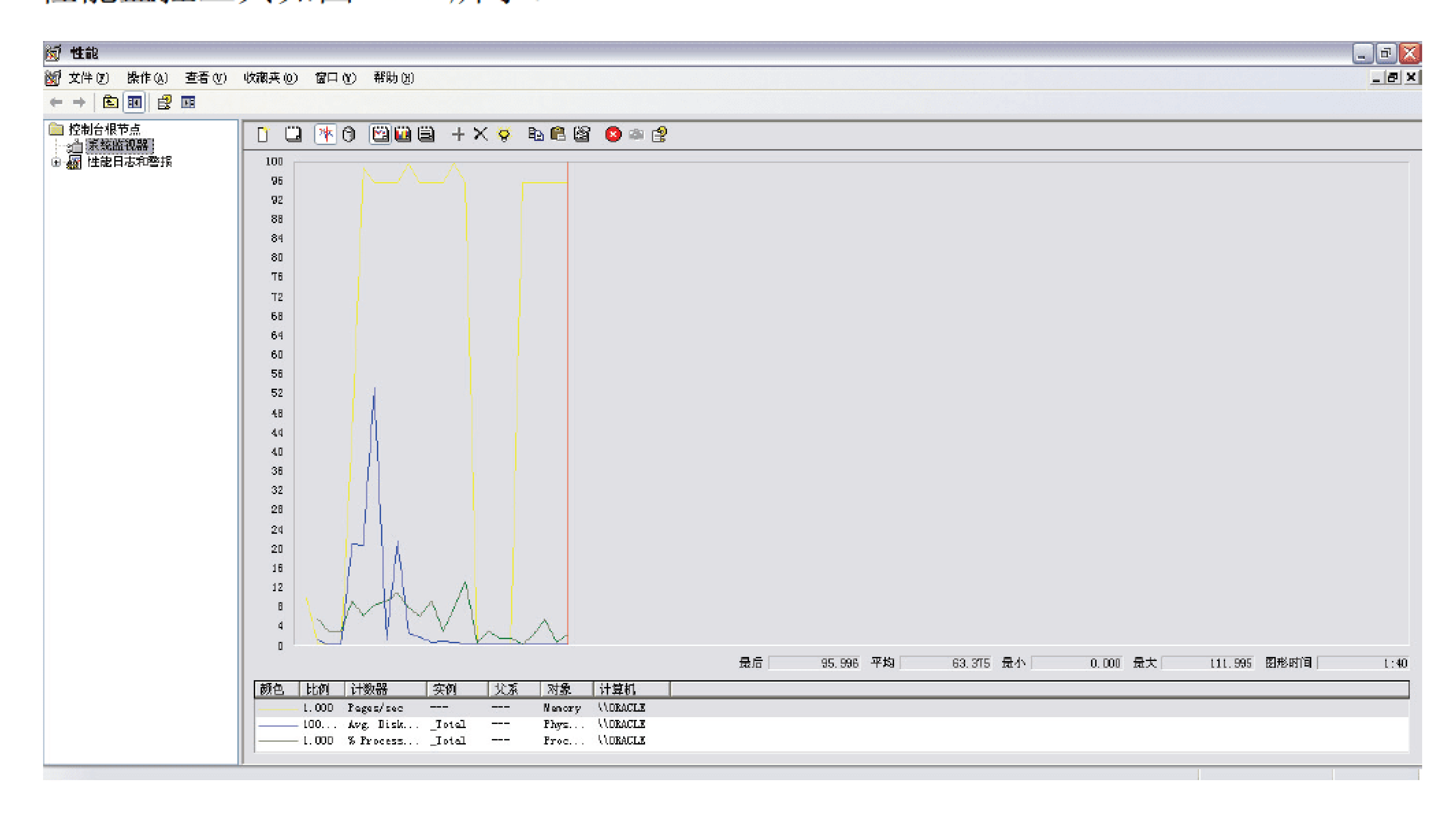


图 15-3

性能监控窗口上是记录 3 台 Tuxedo 服务器内存的分配情况，在上面的例子中，很清楚地看到有一个服务器分配的内存没有被释放。

要配置性能监控器来显示所有名为“simpserv”的进程内存使用，“+”对话框在单击 Add 按钮的之前如图 15-4 所示。

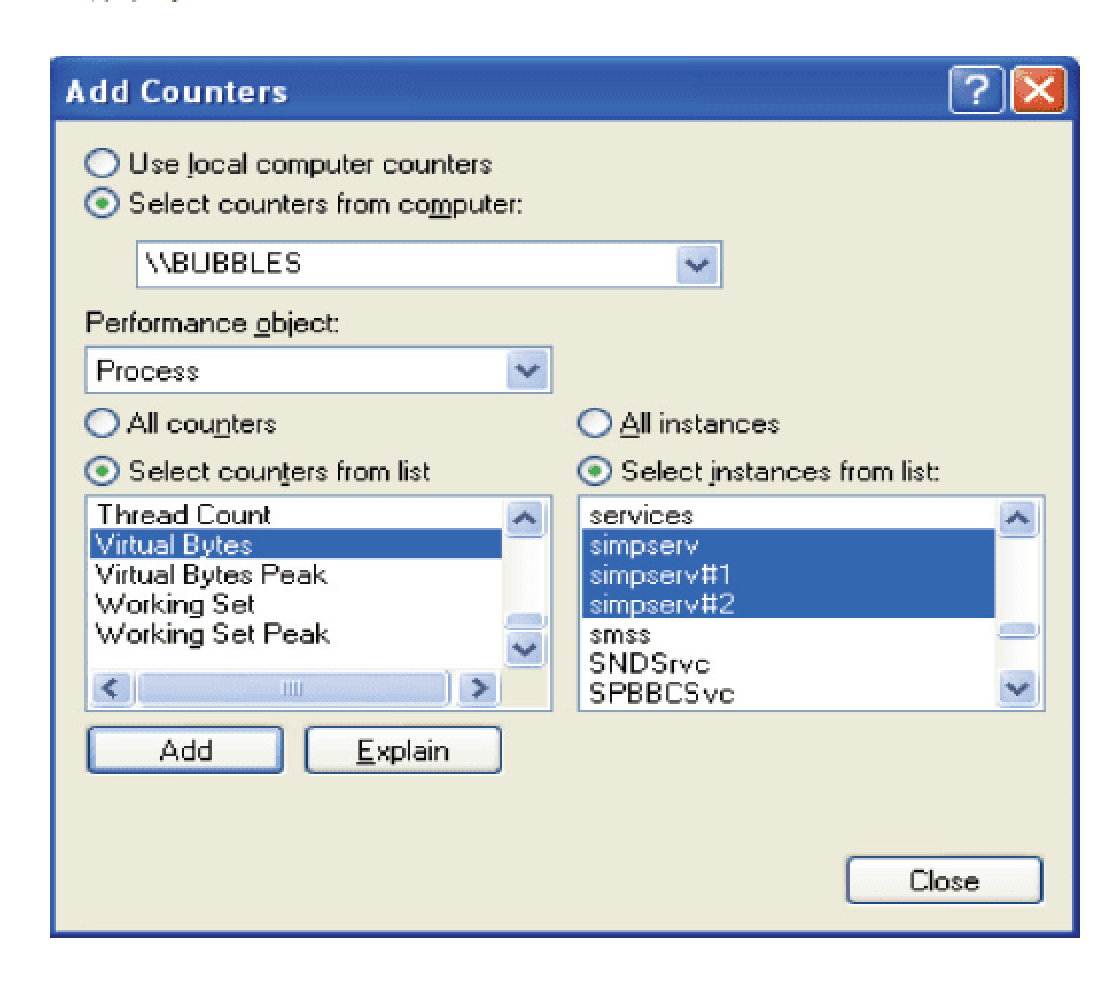


图 15-4

2. UNIX

UNIX 或者 Linux 上，可以使用一系列的工具显示一个进程内存的利用率，这里有 top 和 ps。

（1）top

top 用来实时查看处理器活动的面貌。它显示系统中 CPU 使用最多的进程列表，并且能提供交互式的管理进程的界面。它可以按照 CPU 的使用情况、内存的使用情况以及运行时间对任务进行排序。大多数特征可以通过交互式命令或者指定个人或者系统范围配置文件中的特性。

示例 15-2：

```
37 processes: 36 sleeping, 1 running, 0 zombie, 0 stopped
CPU0 states:  0.0% user,  2.0% system,  0.0% nice, 97.0% idle
CPU1 states:  0.0% user,  0.0% system,  0.0% nice, 100.0% idle
CPU2 states:  1.0% user,  0.0% system,  0.0% nice, 98.0% idle
CPU3 states:  0.0% user,  0.1% system,  0.0% nice, 99.0% idle
Mem:  1030616K av, 1020340K used,   10276K free,     0K shrd,  129420K buff
Swap: 2040212K av,  278396K used, 1761816K free      515496K cached
  PID USER      PRI  NI   SIZE   RSS  SHARE STAT %CPU %MEM  TIME COMMAND
 7400 root       17   0    948   944    756   R   1.9  0.0  0:00 top
 7387 root       11   0   2664  2664   1776   S   0.9  0.2  0:00 sendmail
21906 admin       9   0    216   184    120   S   0.0  0.0  0:15 syslogd
21908 root        8   0    172   116     76   S   0.0  0.0  0:00 crond
21909 daemon      9   0    108    44     32   S   0.0  0.0  0:00 atd
21910 root        9   0    136     4      0   S   0.0  0.0  0:00 su
```

```
21914 mysql      9  0    164     4      0   S   0.0  0.0  0:00 safe mysqld
21916 root       9  0    336   196    140   S   0.0  0.0  0:05 sshd
21951 mysql     13  5  14468  9380   1180 S N   0.0  0.9  0:05 mysqld
21958 mysql     12  5  14468  9380   1180 S N   0.0  0.9  0:08 mysqld
20011 web        9  0  15716   12M   3064   S   0.0  1.2  0:03 httpd
20012 web        9  0  14808   10M   2536   S   0.0  1.0  0:02 httpd
 4012 root       9  0   1940  1928   1320   S   0.0  0.1  0:00 sendmail
 5571 root       9  0   2208  2204   1580   S   0.0  0.2  0:00 sendmail
 5723 root       9  0   2640  2584   1788   S   0.0  0.2  0:00 sendmail
 7352 root       9  0   2192  2192   1568   S   0.0  0.2  0:00 sendmail
 7358 web        9  0   9848  5004   1536   S   0.0  0.4  0:00 httpd
 7365 root       9  0   2728  2728   1792   S   0.0  0.2  0:00 sendmail
 7384 root      10  0   2188  2188   1564   S   0.0  0.2  0:00 sendmail
 7399 landingbj 13  0   1080  1080    876   S   0.0  0.1  0:00 exec.tcl
```

数据的结果会持续刷新直到用户决定退出（按 q 键），默认 top 命令输出的结果是按照 CPU 使用率来排序的。只要在交互模式下，可以修改为按内存利用率排序。排序列制定到运行的 top 的实现，请参考 man 页面。

通过连续监控 Tuxedo 服务进程，可以看出一个进程的内存值是否随着时间的推移不断增长。如果发现分配给一个进程的内存空间不断增长，那么该应用程序可能就存在内存泄漏。

（2）ps

ps 显示进程的运行信息。这些信息是按列排序的，基于一系列的关键词显示。用户需要显示虚拟内存值的关键词是 vsz。注意在 HP 的系统上，需要设置 UNIX95 环境变量才能使用-o 等选项。当 ps 执行时，进程文件系统（procfs（5））应该挂载，否则可能有信息显示不全。

使用 ps 以及 grep 找到 Tuxedo 服务器的进程号来查找用户感兴趣的 Tuxedo 服务器。假设 Tuxedo 服务器执行程序的名称是 simpserv。

这样就会产生输出如下代码：

示例 15-3：

```
landingbj   11148       1 con  14:37:45 simpserv
```

第一列是执行进程用户名称，第二列是执行程序的 id 号，第三列是父进程的 id 号（1 表示这个进程的父进程是 init 进程）。

（3）tmadmin

要获取 Tuxedo 服务器进程的 id，可以使用 tmadmin 控制台工具。把输出设置为 verbose，然后使用 psr 命令。

示例 15-4：

```
C:\temp\simpapp>tmadmin
tmadmin - Copyright (c) 1996-1999 BEA Systems, Inc.
Portions * Copyright 1986-1997 RSA Data Security, Inc.
```

```
    All Rights Reserved.
    Distributed under license by BEA Systems, Inc.
    Tuxedo is a registered trademark.
    > verbose
    Verbose now on.

    > psr -g GROUP1
    Group ID: GROUP1, Server ID: 1
    Machine ID: SITE1
    Process ID: 6156, Request Qaddr: 770, Reply Qaddr: 770
    Server Type: USER
    Prog Name: C:\TEMP\SIMPAPP\simpserv.exe
    Queue Name: 00001.00001
    Options: ( none )
    Generation: 1, Max message type: 1073741824
    Creation time: 6/05/2005 1:59:12 PM
    Up time: 0:49:49
    Requests done: 0
    Load done: 0
    Current Status: (  IDLE )
```

从示例 15-4 可以看到 simpserv 服务器的进程 id 号。

因此可以使用 id 11148，通过 ps 命令取得这个进程的虚拟内存大小。

示例 15-5：

```
    $ ps -p 11148 -o vsz (display the memory usage of the process with process
id 11148) VSZ  1592
```

从示例 15-5 可以看到 simpserv 进程内存大小为 1592KB。如果使用这种方法，可以找到分配给进程的内存的增长情况，然后就可以找到应用中的内存泄漏。

另外一个使用 ps 的例子是通过联合使用脚本语言如 awk 来显示特定的详细信息：

```
    $ ps -elf | awk "{ if ( $10 > 1000 ) print $0; };"
```

此命令输出系统中虚拟内存大于 1000 的进程。注意 ps 的参数并不是对于所有的都成立的规范化参数，在使用之前需要先查看相关平台的 man 手册来决定哪个参数提供虚拟内存信息。

15.4.2　隔离应用程序来跟踪内存泄漏

本节针对了解应用程序并且能够获取其源代码的人员。大多数 Tuxedo 应用程序可以分割为组件部分，通过隔离应用程序的组件部分可以尝试找出导致内存泄漏的代码区。这样可以节省很多时间，但是如果不进行早期合理的规范也不值得再次花费很多时间隔离代码。

15.4.3 隔离应用服务

根据应用的架构，应用程序提供的服务可以分割为几个服务器。一个特定服务器发布的服务，对于判断是应用代码问题还是调用库的问题是非常有用的。

通过在一台服务器内运行目标是特定服务的测试，就能隔离导致服务器虚拟内存使用率增长的服务。当应用开发到单元测试阶段时，就应该对各种服务功能的代码进行测试。使用自动化的测试脚本到在同一服务器上进行多次请求，然后使用监控工具来测试内存使用率是否增长。

15.4.4 隔离应用的组织机构的库/代码

服务器内存泄漏可能并不是由暴露问题的服务代码引起的，而是由应用的共享库引起的。如果发现一些服务器不论是用来做什么工作都存在内存泄漏的问题，那就有可能是这一共享组件（库、函数调用等）导致内存泄漏。

15.5 内存分析工具

如果内存泄漏的原因不容易找出，就可能需要内存分析工具来找到这个问题。有很多可用的内存分析工具可以用来分析应用的内存使用情况。有免费的内存监视工具如 memwatch 或者 ValGrind，也有收费的工具如 Purify 或者 insure++。花费在内存分析工具上越多，得到的功能就越多。

内存分析工具的本质就是在应用以及分配的内存中加了另外一层，在这一层来收集内存信息。比如，memwatch 提供一个 c 语言#define 宏来替代标准的 malloc.h 函数。所以当应用程序调用 malloc 时，memwatch 版本的 malloc 会调用存储在其中的内存分配以及地址的信息，然后再调用底层的 C 语言 malloc 函数来分配内存。通过跟踪内存的分配与回收，memwatch 就可以知道应用何时以及怎样分配内存，然后向开发者指明内存泄漏的方向。

对于 Purify，开发者只要把 Purify 库链接到应用就能自动进行内存跟踪。以下是 4 种不同的内存分析工具，每种都有自己的优劣势。

15.5.1 memwatch

Memwatch 是用 C 语言写的用来检测内存泄漏的一个免费的编程工具。它使用高度可移植的 ANSI C 代码，可以运行在任何拥有 C 编译器的硬件上。但是它主要用来检测和诊断内存泄漏，当然也可以用来分析程序的内存使用情况及提供相关日志的功能。注意，该工具跟 Tuxedo 提供的 tpalloc()调用不兼容。因此该产品对于用户而言是有限制的，因为它无法用于检测发送到其他服务的缓存的分配/回收。

以下是版本 2.71 的一些特性（ANSI C）。

（1）日志文件或者使用 trace()宏的用户函数。

（2）容错，能修复其数据结构。

（3）检测是否两次释放或者多次释放。

（4）检测未释放的内存。

（5）检测上溢与下溢的内存缓存。

（6）可以设置最大分配的内存或者重压测试的应用程序。

（7）Assert()和 verify()宏。

（8）能检测空指针写。

（9）支持操作系统指定的地址验证以避免 GP（分割错误）。

（10）收集应用程序的分配信息、模块及运行级别。

（11）初步支持线程（见 FAQ）。

（12）初步支持 C++（默认关闭，谨慎使用）。

要在代码中使用 memwatch，需要包括一个新的头文件，定义预编译变量，然后重新编译代码。下次在运行执行程序时，memwatch 就能把分析的内容写入到日志文件中，或者调用一个用户自定义宏。

15.5.2　Purify

Purify 是一个软件开发人员用于检测程序的内存访问错误的调试程序，尤其是那些用 c 或者 c++写的程序。Purify 会做一个动态验证，例如，它能发现正在运行的程序的错误，所以它更像一个调试器。相反，静态验证或者静态代码分析是用于检测源代码的错误甚至不用预先编译或者运行，用于发现逻辑上的一致。C 编译器的类型检查是一个静态验证的例子。

当一个程序使用 Purify 链接时，正确的验证代码会通过解析与加入新的对象代码、库自动插入到执行程序。如果发生内存错误，程序会打印出错误的确切位置、参与的内存地址以及其他相关信息。Purify 也包括垃圾收集函数来检测内存泄漏，这个检测可以是在程序退出时或者从一个调试器本身调用泄漏检测函数的任意时刻。

Purify 发现的错误包括以下内容。

（1）数组的读写边界。

（2）试图访问一个未分配的内存。

（3）释放未分配的内存（通常因为释放相同的内存两次）。

值得一提的是，这些错误通常都不会立即导致程序出错，当程序正在运行时不使用公斤，很难检测这些错误，除非注意到不正确的程序的行为引起的错误。因此 Purify 对于告诉程序员错误发生的具体位置有很大的帮助。因为 Purify 是通过检测所有目标代码来工作的，它可以发现发生在第三方或者操作系统内部的错误。这些错误往往是由于程序员传递了不正确参数给库调用，或者误解了协议通过库来释放数据结构引起的。这些错误通常是很难找出并且修复的。可以检测非致命性的错误是 Purify 与其他类似调试器的一个主要区别。Purify 的工作过程是自动进行的。无须安装就能正确地进行编译代码，因此大多数程序员会用到它提供的简单并且强大的自动内存访问验证的功能，至少如果这些在他们系统

上可用时会这么做。

Purify 是专利产品并且对于商业用途的许可是很昂贵的，大学可以获得便宜的学术型的许可文件。

15.5.3 Valgrind

Valgrind 是用于内存调试、内存泄漏检测、剖析的一个免费工具。Valgrind 的初衷是设计一个 Linux x86 平台下 Purify 的免费版本，但后来演变为一个创建动态分析工具的通用框架，如检查和分析。它拥有极佳的口碑并且官方用于 Linux 程序员中。

Valgrind 本质上是一个使用 JIT 编译技术的虚拟机。调试的程序都无法从宿主处理器中直接运行，相反 Valgrind 开始把程序转换到一个临时的更简单的形式，称为 ucode。转换后，这样 Valgrind 可以自由地去对临时 ucode 做处理，然后再将 ucode 转换回到 X86 代码宿主处理器上并运行它。在这种转换过程中，会损失相当一部分性能，通常在 Valgrind 下运行的代码以及工具会比正常情况下慢 4～5 倍。然而 ucode 形式却更好测量，它使其更容易写工具，对于大多数项目而言，在调试过程中这种慢的程度并不是什么大问题。当移动或者操作数据时，测量代码会记录下 A 和 V 位，因此在单一位的级别下它们总是正确的。这里与 Purify 不同，它只能检测没有初始化的内存拷贝，这对于其自身来说是一个合法操作。例如，X Window 系统客户端库频繁拷贝部分未经初始化的结构，从而导致引发大量虚假的警告。迫使程序员关闭警告或者不初始化内存拷贝。

相比 Memcheck 而言，Valgrind 有其他几种工具，例如，Addrcheck 是一个轻量级的 memcheck 姊妹版本，运行更快，所需的内存更少，但是捕捉到更少的几种类型的 bug；Massif 是一个堆分析器；Helgrind，是在多线程代码下能够探测竞争的工具；Cachegrind 是一个缓存分析器。同样也有几种外部开发工具。但是此时，Valgrind 支持的平台只有 Linux x86。当然也有非官方的移植版本，如 freebsd 和移植到 Linux powerpc 上的体验版本。现在还没有移植到 Windows 上的版本，当时有一个体验版本能够与 wine 交互，在 Linux 上调试 Windows 软件。增加平台支持是一个长远的目标，但是需要做更多的工作。

15.5.4 Insure++

Insure++是一个计算机程序的内存调试器。软件开发者使用它来探测 c 和 c++的各种错误。它是由 parasoft 公司开发的，并且与其他内存调试器功能相似，如 Purify 与 Valgrind。Insure++能够自动查找，释放内存的各种访问方式，数组越界、释放未分配的内存（通常是程序员释放内存两次或者释放全局或堆栈内存）。和 Purify 及 Valgrind 不同的是，insure++在源代码级别插入性能测试的代码，这样就允许它探测其他工具缺失的错误。特别是 insure++能探测自动化数组的缓存溢出，以及指针意外地从一个有限的地址区域跳到另外一个区域。

比如下面的代码例子。

示例 15-6：

```
#include <stdlib.h>
```

```
int main()
{
    char *p = malloc(1024); /* first dynamically-allocated block */
    char *q = malloc(1024); /* second block */
    p += 1200; /* at this point, "p" is very likely to point into the
                  second block, but depending on that would be a mistake */
    *p = 'a';  /* invalid write (past the end of the first block) */
    return 0;
}
```

15.6　常见的内存泄漏的原因

以下依次说明引起 Tuxedo 应用内存泄漏的常见问题。

15.6.1　非成对使用 tpalloc()/malloc()与 tpfree()/free()

有时候，开发人员会在完成工作之前忘记释放预先分配的内存，特别是遇到一个错误的时候。

请看以下代码。

示例 15-7：

```
ServiceA(TPSVCINFO *transb){
    char *buffer;
    buffer = malloc(1024);        (1)

    /* do something */
    if( error occurs){
       tpreturn(TPFAIL,….);       (2)
    }

    free(buffer);                 (3)
    tpreturn(TPSUCCESS,….);
}
```

这里，在（2）处分配的内存 buffer 会发生泄漏。这是因为如果处理服务的过程中发生错误，在（1）处分配的内存没有得到释放。如果处理的过程中没有发生错误，就会运行到（3）处，在这（1）处分配的内存就会被释放。这里需要强调的是当给 Tuxedo 应用分配内存时，需要特别慎重对待，特别是在发生错误的条件下。需要确保任何之前分配的缓存在发生错误返回时都得到释放。

15.6.2　重写指针

在某些情况下，包含分配内存地址的指针会被其他变量入侵而损坏。

看一下下面的代码。

示例 15-8：

```
ServiceB(TPSVCINFO *transb){
    char buffer1[100];
    char *buffer2;
    char buffer3[100];

    buffer2 = malloc(100);          (1)
    memset(buffer1,0x00,200);       (2)

    /* do something */
    free(buffer2);                  (3)
    …
}
```

在（1）处，buffer2 指向分配的内存。在（2）处 buffer2 变量中的地址被 memset()调用的入侵而重置了。buffer1 只有 100 字节的内存，但是 memset()调用则设置了 200 字节的连续内存。这就造成了 buffer2 变量包含的内存地址的重写。

因此，当需要时，请谨慎使用 memset()或者 strcpy()。

15.6.3 C 库函数的 bug

这样的情况不经常发生，但是有时也遇到 C 库函数导致内存泄漏。特别是 ascftime()函数在某些场合可能导致内存泄漏。

这里最佳的途径是当确定代码没有内存泄漏，但是却仍然有内存泄漏发生，如果可能，可以把这个问题隔离到单独一台服务器，然后试图逐一调用服务来判断哪个服务正在泄漏内存。

一旦服务记录下来，就需要使用重复的方式来移除源码中的行，直到找到产生问题的特定行的代码。一种处理的方式是移动 tpreturn()调用的位置。如果 tpreturn()在服务的末尾处调用，把它移到服务的中间来。用户不关心服务如何处理这个任务，只关心其内存泄漏。如果这样内存不再泄漏，则说明内存泄漏发生在服务的第二部分，否则就是第一部分。继续执行这个步骤，直到能确定源码中内存泄漏的准确行。

第 16 章　与全局事务 XA 相关的故障

16.1　问题描述

Tuxedo 全局事务处理的问题分为以下两种。

第一种是配置问题，因为配置 Tuxedo 系统支持 XA 资源有时比较复杂。

第二种是事务运行时的问题，运行环境的复杂性，如使用多种资源管理器，导致发生的问题也相对复杂。

16.2　通过配置让 Tuxedo 支持事务

下面是 Tuxedo 支持 XA 事务的配置启动过程。按照以下步骤配置，系统就会按照预定的方式工作。

（1）配置 Tuxedo 支持事务。

（2）为欲连接的数据库建立一个 TMS 服务器。

（3）使用 XA 库编译 Tuxedo 服务器来连接数据库。

（4）把 Tuxedo 服务器绑定到同一个 TMS 服务器组，并配置 TMS 服务器、OPENINFO 字符串以及 TMS 服务器数量。

（5）启动系统并确认所有服务器按照预期方式启动。

16.2.1　配置 Tuxedo XA

为了支持事务，需要对 UBBCONFIG 文件作相应的更改。

总结相关更改信息见表 16-1。

表 16-1

更改要求	例子
在 UBBCONFIG 的*RESOURCES 段通过更改 MAXGTT 值来增加事务并发数	*RESOURCE #Increase the MAXGTT from the default of 100 to 255
事务的超时和在提交时事务对调用者的响应的方式可以在*RESOURCES 段进行修改	MAXGTT 255
通过 tmadmin 和 crdl 命令建立设备供 TLOG 使用。设备必须在域的每台机器上都要建立	tmadmin -c crdl -z D:\myapp\TLOG -b 1000
为指定名字或默认的机器名创建 DTP 事务日志	tmadmin crlog -m SITE1

续表

更改要求	例子
在 UBBCONFIG 的*MACHINES 段，指定本节点的 TLOG 的路径	*MACHINES TESTMACHINE1 TLOGNAME=TLOG TLOGDEVICE=D:\myapp\TLOG
修改 UBBCONFIG 的*GROUPS 段来添加指定 TMS，并且为这个 TMS 的 RM 设置 OPENINFO 的值	*GROUPS BANKB1 GRPNO=1 TMSNAME=TMS_SQL TMSCOUNT=2 OPENINFO="Tuxedo/SQL:*APPDIR*/bankdl1:bankdb:readwrite"
启用 XA 的服务器必须与正在使用的资源管理器配置到同一组	*SERVERS MYSERVER GROUP=BANKB1
在 UBBCONFIG 的*RESOURCES 段通过更改 MAXGTT 值来增加事务并发数	*RESOURCE #Increase the MAXGTT from the default of 100 to 255 MAXGTT 255
事务的超时和在提交时事务对调用者的响应的方式可以在*RESOURCES 段进行修改	

那么如何决定 TLOG 与 MAXGTT 大小呢？

关于 TLOG 与 MAXGTT 的大小，尽管它们间是相关联的参数，但它们却是不同的东西。它们在 UBBCONFIG 中定义范围如下。

0 <= MAXGTT <= 2048　　Tuxedo 5.x 以前

0 <= MAXGTT <= 32768　　Tuxedo 6.x 以后

0 <= TLOGSIZE <= 2048

TLOG 为每个预提交事务存储一个页。它必须与机器上通过两阶段提交预备进行协调的并发事务数的大小一样。在 UBBCONFIG 中通过 TLOGSIZE 来设置 TLOG 的大小，然后添加一个附加信息（以页面大小为基础），最终通过 tmadmin 创建 TLOG 文件（TLOG 大小=TLOGSIZE+overhead（20%～30%）。

TLOG 的页面大小各不相同，大多数机器都是 512，但 RS60000 是 4096，HP 重新发布的 5.0 二进制文件为 1024。为了能够恢复事务，TLOG 文件应该有原始和镜像磁盘。如果失去了所有的 TLOG，这是无法恢复的。一旦事务参与者反馈已经提交，TLOG 条目就可以访问了。

全局事务表（GTT）是 BB 中的一张表，它会跟踪机器中已经开始的全局事务。如果机器是事务的发起者或参加了全局事务，它就需要一个 GTT 条目。这个表的大小由 MAXGTT 参数控制。GTT 中的条目在事务提交后就会被释放。正因为如此，GTT 可能需要比 TLOG 大。

16.2.2　创建事务管理器和 XA 服务器

要使一个资源管理器（RM）如一个数据库能和 Tuxedo 协同工作，RM 的提供商必须提供支持 XA 协议的一系列的库。这些 XA 的库不仅要链接到作出调用的 Tuxedo 服务器，

而且需要链接到一个特殊的 Tuxedo SERVER，称为事务管理服务器（TMS）。XA 库可用于所有主流数据库和其他中间件产品。

为了创建一个 TMS，Tuxedo 提供创建 TMS 的工具称为 buildtms。buildtms 工具使用标准的方法创建管理器服务进程，并且与指定的 RM 库合并以构建一个特定的 TMS。为了达到这个目标，buildtms 工具需要知道链接哪些 XA 库，这些信息来自 RM 文件。

在 RM 文件中包含所有常见的 XA 资源管理器。该文件位于$TUXDIR/udataob 目录，为文本文件。每个条目包含一系列的属性并提供了一个 RM 名称（第一个字段）与 xa_switch_t 结构体名称（第二个字段）以及提供 RM、XA 实现的一系列库。

在 RM 文件中的第 3 个字段是最关键的，并且通常导致最多问题。因为这个部分的库用来创建特定 RM 的 TMS，并且这些库将被绑定到任何希望在事务中涉及到的 Tuxedo 服务器。

下面的例子展示了 RM 文件中一个简单的字符串用于连接到 AIX 上的 DB2 数据库。

在这个例子中，UDB_XA 是为 DB2 定义的 RM 的名称，db2xa_switch 是 DB2 为类型 xa_switch_t 结构所定义的名称。

在文件${TUXDIR}/udataobj/RM 中添加如下定义。

示例 16-1：

```
# DB2 UDB
UDB_XA:db2xa_switch:-L${DB2DIR} /lib -ldb2
```

一旦 RM 文件已经设置完成，如果希望使用这个 RM 就必须使用 buildtms 创建 TMS 文件，并且安装在$APPDIR 下。

以下例子将为 DB2 创建一个 TMS。

示例 16-2：

```
buildtms -o $APPDIR/TMS_UDB -r DB2_XA
```

新生成的 TMS 执行文件名称为 TMS_UDB。这个名称必须配置到 Tuxedo UBBCONFIG 文件的*GROUPS 下面。

除了给将使用的数据库建立一个 TMS 服务器，有必要建立一个连接 XA 库版本的 Tuxedo 服务器。创建这样一个服务器，要使用 buildserver –r。需要特别注意的是 buildtms 定义的 RM 和 buildserver 定义的 RM 必须相同，这样才能让 XA 的库正常工作。

创建 TMS 服务器遇到的最常见问题是每种数据库都需要包括什么库。默认的 RM 文件只定义了几个常用资源管理器。通常需要通过扩展来支持新的数据库。这些链接到 TMS 的库往往因为数据库的版本不同而变化多端。因此检查并确保使用的 RM 文件中包含正确的库是很重要的。

16.2.3　XA-OPENINFO 字符串

当在使用 Tuxedo 中支持 XA 的服务器时，需要传递给 OPENINFO 参数一个字符串，告诉 Tuxedo 服务器如何连接后端资源（如数据库）。每个字符串的值包含的是需要打开一

个 RM 的信息，由 RM 提供商传递给这些参数一些适当的值。

例如，如果使用 ORACLE 作为 RM，需要提供的以下项中的显示的值。

示例 16-3：

```
OPENINFO="ORACLE_XA:Oracle_XA+Acc=P/Scott/Tiger+SesTm=30+LogDit=/tmp"
```

OPENINFO 字符串对各种数据库都不同。实际 OPENINFO 字符串取决于数据库提供商。因此如果决定实际所需的 OPENINFO 字符串，需要查看数据库提供商的文档。

如何在 UBBCONFIG 中加密 ORACLE OPENINFO 的密码呢？

在 OPENINFO 字符串中传递给一个 RM 的密码既可以存储为纯文本也可以存储为加密的形式。要加密密码，首先在 OPENINFO 字符串中输入 5 个或者更多连续的星号。

示例 16-4：

```
OPENINFO="Oracle_XA: Oracle_XA+Acc=P/Scott/*****+SesTm=30+LogDit=/tmp"
```

然后通过 tmloadcf 来加载 UBBCONFIG 文件。当 tmloadcf 遇到了星号的字符串时，它会提示输入一个密码。

示例 16-5：

```
tmloadcf -y /usr5/apps/bankapp/myUBBconfig
Password for OPENINFO (SRVGRP=BANKB3):
Password
```

tmloadcf 以加密的形式将密码存储到 TUXCONFIG 文件。如果使用 tmunloadcf 从 TUXCONFIG 文件来获取信息重新生成 UBBCONFIG，密码会在重新生成的 UBBCONGIF 文件中以@@作为分隔的加密形式打印输出。

示例 16-6：

```
Oracle_XA+Acc="P/Scott/@@A0986F7733D4@@+SesTm=30+LogDit=/tmp"
```

当 tmloadcf 加载 tmunloadcf 生成的 UBBCONFIG 中遇到以上加密形式的密码，它不需要提示用户再输入一个密码。

16.2.4 TMS 服务器

一旦 TMS 服务器创建好了，下一步就是把 TMS 服务器包括在 UBBCONFIG 文件中并与将调用 RM 的实际服务器放在相同的组中。

同一个组的 TMS 服务器数量是通过 TMSCOUNT 属性设置的。默认运行 TMS 服务器的数量是 3 个，且设置值不能低于 2 个。不可能在一个组中运行单一的 TMS 服务器，因为这可能导致出现问题。最好运行两到三个 TMS 服务器，并随时监视系统。

如果遇到以下情况，可能需要增加 TMSCOUNT 的值。

（1）使用“tmadmin psr”，TMS 的“当前服务”的值显示为一个服务名而非“空闲”（IDLE）。

（2）从 UNIX 的“ipcs -aq”来获取 TMS 消息队列 CBYTES 值在多数情况下非常接近

于 BYTE 值。其中，“CBYTE”是在特定消息队列中未完成的字节数，而“QBYTE”是在一个特定的消息队列中允许的字节数。

16.3　运行时问题

16.3.1　调用 tx_open() 或 tpopen() 失败

当一个 Tuxedo 使用 XA 时，通常在服务器初始化时，即在 tpsvrinit()函数中，调用 tx_open()或 tpopne()的 API，建立服务器与 RM 的连接。

如果在 ULOG 中见到下面的错误。

示例 16-7：

```
    LIBTUX-6205 WARN: Server initialization function did not call tx open() or
tpopen() or this call failed
```

这说明开发人员并没有在他们的代码中调用 tx_open()/tpopen()，或调用未成功。

首先得确保开发人员做了正确的调用。如果调用已经检查完毕，再检查 ULOG 中其他信息，并检查 OPENINFO 字符串，以确保这些细节都准确无误。

这两点可能是服务器无法正确建立与 RM 连接的原因。

16.3.2　启发式失败

在正常的事务操作中可能无法完成的因素有数据库或网络故障，甚至是代码中的问题。通常数据库失败不是一个问题（除非它们的数量很大并且能够引起一些问题），但它可能为 Tuxedo 系统造成一个称为启发式失败的错误，这个错误通常需要被关注。

当 Tuxedo 系统发生启发式错误时，系统无法确定事务完成与否。发生启发式错误，Tuxedo 系统在 ULOG 文件中会写入错误信息。然后由管理员（通常是指 DBA 和开发人员）判断在这一时刻发生了什么样的错误以及如何解决这个问题。

启发式错误一旦发生，并没有简单的解决方法。唯一有效的解决方法就是手动检查参与事务中的资源并尝试判断数据库是否正确提交了。

16.3.3　xa_start ()返回 XAER_RMERR

如果 ULOG 中出现以下错误信息。

示例 16-8：

```
    xa start returns XAER RMERR ( -3, a resource manager error occurred in the
transaction branch)
```

这个错误可能有以下几种情况。

（1）TLOG 没有正确创建：需要验证 TLOG 已经被有效创建。

（2）数据库无法连接或脱机：与 DBA 确认该数据库已经启动并且应用程序可以正常的访问到它。

（3）Oralce 的系统表 dba_pending_transaction 没有授予给公用账户 public 查询权限：该 dba_pending_transactions 表必须赋予 public 查询权限。

16.3.4 xa_start()= -9 问题

官方关于 xa_start= -9 错误的解释如下。

示例 16-9：

```
    #define XAER_OUTSIDE    -9  /* resource manager doing work outside global
transaction */
```

表面上是说，在准备启动全局事务时，RM 已经在做外面其他工作了。而实际上，这是由两种原因造成的错误。

（1）在全局事务开始前，本地事务没有提交或回滚，如示例 16-10。

示例 16-10：

```
insert/delete/update SQLs
tpbegin()
```

（2）全局事务的超时，没有得到正确处理。

尽管所有的可写 sql 操作应该放在 tpbegin()、tpcommit()、tpabort()之内，如果全局事务超时并且 sql 的错误没有被正确地处理，也会产生 xa_start=–9 的错误，如示例 16-11。

示例 16-11：

```
tpbegin()
    sql 1    <--- (timeout)
    sql 2
    sql 3
tpcommit()/tpabort()
```

在示例 16-11 中，sql 2 会给出 sqlcode < 0，sql 3 的结果输出在本地事务中。

下面是尽大可能操作，以便找到引起该错误的进程和本地事务的 sql 语句，但由于时效性的关系，有可能拿不到当时的准确信息（这里以 Oracle 数据库为例子）。

（1）登录。

示例 16-12：

```
sqlplus sys/chang_on_install(password..)
```

（2）检查长期本地事务。

示例 16-13：

```
select xidusn, start_time, ses_addr from v$transaction;
```

试图查找较早的 start_time。

（3）查找进程。

示例 16-14：

```
select sid,program, sql_address from v$session where saddr=%ses_addr%;
```

此处的%ses_addr%是从上次 select 语句中选择的数值。

（4）在本地事务中查找 sql 语句的结果。

示例 16-15：

```
select sql_text from v$sqltext where address='%sql_address%';
```

此处的%sql_addr%是从上次 select 语句中选择的数值。

16.3.5　Oracle TMS 挂起错误

有时 Tuxedo 的 TMS 挂起或 TMS 的队列长度增加很快。

该问题的解决一般分为两步完成。

1．检查 3 种超时的设置

（1）Tpbegin(T1)中的全局事务超时 T1。

（2）OPENINFO 中的会话空闲超时 SesTm=T2。

（3）Oracle 系统全局事务锁超时_distributed_lock_timeout=T3，在 init.ora 中。



确保满足 T1<T2<T3。

2．增加 Oracle 内核参数

比如，在 init.ora 中设置 max_commit_propagation_delay≥90000（需要和 DBA 确认）

有时 Oracle 的 Bug 也会造成 TMS 挂起问题。

16.4　XA 跟踪

16.4.1　TMTRACE

运行时跟踪功能 tmtrace，允许应用程序管理员和开发人员跟踪一个 Tuxedo 应用程序的执行。

要跟踪 XA 的调用，在执行 tmboot 前先设置 TMTRACE=xa:ulog:dye。

在运行时跟踪的一个服务器或一个组的行为也可以使用 tmadmin / changetrace 动态修

改，如下面的示例。

示例 16-16：

```
$tmadmin
>chtr -g G_YZ -i 30001 xa:ulog:dye
```

G_YZ 是组的 ID，30001 是服务器的 ID。

16.4.2 DbgFl

与此对应，Oracle XA 跟踪的收集是通过添加一个 debug 标志到 XA 的 OPENINFO 的跟踪。

示例 16-17：

```
+DbgFl=<level>

Trace Levels
============

0x1     Trace all XA calls
0x2     Trace non-XA interface calls
0x4     Trace intermediate steps
0x8     Trace Oracle XA version
0x20     If mode=ansi closes cursors prior to detaching in Version 8. For
details see [BUG:2622558]
```

这些可以连接起来，“+DbgFl=15”可以设置前 4 个跟踪级别。

要启用 Tuxedo 中的 XA 跟踪，需要更改 UBBCONFIG 中 OPENINFO 的参数。

示例 16-18：

```
OPENINFO="ORA734d:Oracle XA+Acc=P/system/manager+Logdir=/opt/tuxedo/sam
ple/landingbj+Sestm=100+DbgFl=0x7"
```

XA 跟踪通常是通过 DbgFl 参数设置，如果移除它，跟踪就会关闭。

示例 16-19：

```
OPENINFO="ORA734d:Oracle_XA+7Acc=P/system/manager+Logdir=/opt/tuxedo/sa
mple/landingbj+Sestm=100"
```

下面给出一个跟踪的范例。

示例 16-20：

```
XA Trace Example
================
The file is stored in whatever +LogDir=<directory> is set to.
A new file is created every day to include the date.
```

```
    135202.1907.0:
    xaostart:xid=0x28-707573743839000000000000000000000000000000000000000
000000000
    000004800138b7caaf0000052b-0000057800000258, rmid=0, flags=0x0

    135202.1907.0: fmt=40, tln=44, bln=8, tid=0x40043a38, bid=0x40043a64

    135202.1907.0: xaostart: upixast completed

    135202.1907.0: xaostart: return XA_OK
```

第 17 章　IPC 相关故障

17.1　Tuxedo 使用的 IPC

Tuxedo 系统大量使用进程间通信（IPC）的资源，以提高通信的有效性以及管理访问共享资源，如公告板（BB）。

IPC 是 UNIX 操作系统核心的一部分，因此所有的 UNIX 都提供了 IPC 的功能。对于 Windows，Tuxedo 开发了 IPC 模块，来提供所需的功能。

由于 Tuxedo 广泛使用 IPC，因而通常需要调整操作系统的 IPC 值，从而支持 Tuxedo 的高效运行。Tuxedo 常用的 IPC 资源有 3 类，它们是信号量、消息队列和共享内存。在对 IPC 进行调整之前，先分别了解一下每一类 IPC 资源的功能和特性。

注：只有少数操作系统（如 AIX）的 IPC 资源是自适应的，不需要手工调整，其他大多数的 UNIX 系统（如 Solaris、HP-UX 等）以及 Linux 的 IPC 资源需要手工调整，否则会导致 Tuxedo 系统无法运行或效率低下。

17.1.1　信号量（Semaphore）

信号量是在操作系统级别设置的，提供锁机制（类似于文件锁）的一系列标志。当一个资源被信号量锁定后，可以防止操作系统中的其他程序访问该资源。可以通过它们来控制文件、共享内存、或其他任何需要限制访问的资源。

一个共享资源通常通过一个信号量来进行资源的保护。该信号量的初始值为“1”，表明该资源可用。一个进程访问这个资源之前，它先检查信号量的值。如果值是“1”，则立即将它置为“0”（说明该资源被占用），接着便可以对此资源执行操作，没有其他任何进程可以同时访问。如果信号量为“0”，其他进程查询或修改资源之前，必须等待。当完成资源的访问时，进程负责把信号量置为“1”，说明此进程访问资源结束，从而允许其他进程继续访问。

一个信号量的基本功能是，它可以被设置，被检查（是否已经设置），也可以等待，直到信号被清除并重新设置。

信号量一个复杂之处在于当进行信号量编程时，申请的不是单一的信号量，而是一个信号量集。用户可以申请只包括一个信号量的信号量集，而 Tuxedo 不是这么做，它会在建立一个信号量集的同时申请大量的信号量。Tuxedo 用信号量来确保对共享资源访问的一致性。信号量配置太低会导致 Tuxedo 系统应用程序无法启动。

17.1.2　消息队列（Message Queue）

消息队列提供了系统中不同的进程之间传递信息的途径。消息队列可以在进程之间共

享，一个进程放在队列中的消息可以被其他进程读取。

Tuxedo 系统使用 UNIX 系统消息队列进行客户端与服务端的通信。这类典型的消息包括服务请求、服务应答、会话消息、通知消息、管理消息、事务控制消息等。

每一个 MSSQ 集（多服务单队列）和每个单独的服务器都有一个接收请求消息队列。每个客户端都有自己的应答队列。服务器若指定 REPLYQ 参数，则也拥有自己的应答队列。

对于正确调优一个应用程序来说，调整内核的消息队列参数是至关重要的，不正确的参数值可以导致无法启动，或严重的性能下降。

17.1.3 共享内存（Shared Memory）

共享内存是操作系统中一个或者多个进程之间共享的一段内存。多个进程可以直接读写共享内存，所以是最快的一种进程间通信机制。为了实现安全通信，它往往与其他通信机制，如信号量结合使用，来达到进程间的同步及互斥。

在 Tuxedo 的环境中，共享内存被用于公告板（BB）和工作站的监听器进程（WSL）对照表。应用程序同样也可以使用共享内存达到自己的目的。

17.1.4 Tuxedo 使用的 IPC 资源

Tuxedo 广泛地使用操作系统的 IPC 资源。

把每个信息张贴到公告板（BB）进程或线程都需要一个 UNIX 信号，这包括本地客户端、服务进程、WSH、BBL、WSL、TMS、BRIDGE，域网关和任何其他系统进程，如 Event Broker 或/Q 进程。

UNIX 信号量是 Tuxedo 系统启动时分配的，因此以下情况一般比较容易识别，即如果操作系统中信号量太少系统没有正常启动就会抛出一个错误，表示没有足够的信号量。

消息队列是 Tuxedo 动态分配的，每个本地客户端和 WSH 都需要有一个应答的消息队列。每个 SERVER 都需要一个请求队列，但 SERVER 可以使用 RQADDR 参数（MSSQ）共享请求队列。SERVER 同样可以设置 REPLYQ = Y，以拥有单独的应答队列。

公告板（BB）使用共享内存，它的大小由 UBBCONFIG 中不同的参数控制，如 MAXACCESSERS、MAXGTT、MAXSERVERS、MAXSERVICES 等。

17.1.5 定义 IPC 限制

为加快处理速度，大多数 Bulletin Board 的空间都是静态分配的，因此正确地对其进行调整是十分重要的。如果它设置太大，内存和 IPC 资源就会过度消耗，如果它设置太小，达到限制时，应用就会失败。

以下是在 UBBCONFIG 的*RESOUECES 段下影响共享内存大小的参数。

（1）MAXACCESSERS：一个节点允许连接到 Tuxedo 系统的进程的最大数。它不是所有进程的总和，而是同时访问这个节点的最大进程数，默认值为 50。（可以在每台机器的 MACHINES 段设置 MAXACCESSERS 覆盖默认值）。

（2）MAXSERVERS：应用中的服务器进程的最大数量，它包括所有管理服务器（如 BBL、TMS），它是应用中所有进程的总和，默认值为 50。

（3）MAXSERVICES：应用发布的服务的最大数量。它是系统中所有服务的总和，默认值是 100。

（4）MAXGTT：应用支持的全局事务的最大数量，默认值是 100。

（5）MAXINTERFACES：在 Tuxedo 应用中指定 CORBA 接口的最大数量。

（6）MAXCONV：指定一台机器上同时进行的会话的最大数量。

17.2 IPC 设置

通常在 Tuxedo 中 IPC 故障的分析相对较简单，它会作为 Tuxedo 错误日志存储到 ULOG 中。不过，Tuxedo 系统的错误不仅可能产生在启动时，还可能出现在运行状态。

因此解决 IPC 的问题需要从两个方面入手。

首先，应配置操作系统的预期处理 IPC 的负载。

其次，如果 IPC 的环境配置出现问题，需重新配置操作系统的环境，以提高 IPC 处理能力。

注意在操作系统中，Tuxedo 不是唯一消耗 IPC 资源的应用程序，这是十分重要的。其他应用程序（如 ORACLE 数据库），或 Tuxedo 其他应用程序的实例，也需要使用 IPC 资源，所以要设置 IPC 的资源，以容纳操作系统上所有的应用程序。

要确定一个 Tuxedo 应用需要占用的 IPC 资源大小，可以运行以下命令。

示例 17-1：

```
tmloadcf -c [ubbconfig file]
```

运行此命令将出现下面的结果。

示例 17-2：

```
Ipc sizing (minimum /T values only) ...
Fixed Minimums Per Processor
SHMMIN: 1
SHMALL: 1
SEMMAP: SEMMNI
Variable Minimums Per Processor
                        SEMUME,           A                           SHMMAX
                        SEMMNU,           *                             *
Node                    SEMMNS  SEMMSL  SEMMSL  SEMMNI  MSGMNI  MSGMAP  SHMSEG
------                  ------ ------  ------  ------  ------  ------  ------
PRODUCTIONMACH            105      13     100   A + 1      31      62    173K
where 1 <= A <= 8.

The number of expected application clients per processor should be added
to each MSGMNI value.
```

其中，A 表示一个变量，它的值介于 1 和 8 之间，首先要确定“A”的值，用“A*SEMMSL”除以 SEMMSL 值，得到的就是 A 的值。一旦得到 A 的值，就能通过 A+1 确定 SEMMNI 的值。

列出的值都是在操作系统中，运行这个应用必须的最小值。设置操作系统参数时，还需要考虑其他的产品和应用。

下表是操作系统 IPC 内核参数的含义，以及 Tuxedo 应用系统所需的配置。

表 17-1

类型	名字	描述	取值
共享内存	SHMMAX	单个共享内存段的最大尺寸	需大于 BB 的大小，参考 tmloadcf -c 的输出
	SHMSEG	一个进程可用的共享内存段的最大数目	每个进程可使用的最大共享内存量=SHMMAX ×SHMSEG
	SHMMNI	系统范围内允许的共享内存段的最大数目	至少大于 SHMSEG
	SHMMIN	单个共享内存段的最小尺寸	一般设置为 1
信号量	SEMMNS	系统范围的最大信号量数量	至少大于 MAXACCESSERS-MAXWSCLIENTS+13
	SEMMNI	可被创建的信号集的数目	通常等于 SEMMNS
	SEMMSL	每套信号集允许的最大信号量数量	通常等于 SEMMNS，因为 Tuxedo 不会大量创建信号集，但它对每个信号集分配尽可能多的信号量
	SEMMAP	用于管理信号量的映射表大小	SEMMNI+2
	SEMMNU	Undo 结构的数目	通常等于 SEMMNS
	SEMUME	每个进程最大 Undo 数量	通常等于 SEMMNS
消息队列	MSGTQL	操作系统内等待处理的最大消息数	需根据应用系统容量设置
	MSGMNB	消息队列的最大尺寸	至少等于 MSGMAX，Tuxedo 为避免消息队列阻塞，当消息长度大于 MSGMNB ×75%时，消息将被存到文件中去处理，这将大大降低总体性能。
	MSGMAX	每条消息的最大尺寸	至少大于 MSGMNB ×75%
	MSGSEG	系统拥有消息段数目	整个系统消息队列占用的最大空间为 MSGSEG×MSGSSZ，因此 MSGSEG×MSGSSZ 至少要大于 MSGMAX
	MSGSSZ	每个消息段的尺寸	一般设置为 8,16, 32 或 64 字节
	MSGMNI	最多可被创建的消息队列数目	至少大于 MAXACCESSERS+（有响应队列的服务器数-MSSQ 中的服务器数）+MSSQ 的个数+7
	MSGMAP	用于管理消息的映射表大小	MSGTQL + 2

17.3　IPC 命令

下面逐一介绍 IPC 常用命令，可进行 IPC 查看和操作。

17.3.1 ipcs

ipcs 打印 IPC 状态，通过选项来控制输出的信息。

如果没有选项，将打印系统中消息队列，共享内存及信号量的相关信息。

常用参数见表 17-2。

表 17-2

选项	描述
-b	打印允许的最大消息字节数、共享内存大小和信号集中的信号量个数
-o	打印消息队列中的消息数和连接到共享内存的进程数
-q	打印活动消息队列信息
-s	打印活动信号的信息
-m	打印活动的共享内存段的信息

下面是“ipcs –boq”命令行。

“-b”显示消息队列上允许的最大消息的字节数。

“-o”显示队列上消息数量。

“-q”指定只显示队列相关信息（即不显示共享内存与信号量的信息）。

示例 17-3：

```
IPCS status from BEA_segV8.1 as of Wed Jun 22 10:41:54 2005
T    ID     KEY          MODE        OWNER    GROUP  CBYTES  QNUM QBYTES
Message Queues:
q    513  0x0000bea2  -Rrw-rw-rw-        0         0       0      0  65536
q    770  0x00000000  -Rrw-rw-rw-        0         0       0      0  65536
q    515  0x00000000  -Rrw-rw-rw-        0         0       0      0  65536
q      4  0x00000000  --rw-rw-rw-        0         0   11396     37  65536
```

从输出结果可以看到，存在一个 ID 为 4 的队列，有 37 个消息等待处理，共 11396 个字节。

MODE 列显示的是 UNIX 的队列权限，OWNER 和 GROUP 列表示队列组和拥有者。

17.3.2 ipcrm

ipcrm 删除一个或多个消息队列、信号量或共享内存。正常情况下， Tuxedo 应用关闭时自动释放 IPC 资源。这个命令是用来清理 Tuxedo 进程失败后遗留的 IPC 资源。某些情况下，Tuxedo 的服务器没有获得释放 IPC 的资源机会，必须手工操作。

ipcrm 使用的 ID 为 ipcs 命令输出的 ID，可以针对特定的 IPC 资源进行清理。

常用参数见表 17-3。

表 17-3

选项	描述
-m <ID>	根据 id 删除共享内存段
-s <ID>	根据 id 删除信号量
-q <ID>	根据 id 删除消息队列

ipcrm 删除上述所有队列信息。

示例 17-4：

```
ipcrm -q 513 -q 770 -q 515 -q 4
```

17.3.3　tmipcrm

tmipcrm 清除 Tuxedo ATMI 应用程序中所分配的 IPC 资源，如共享内存、消息队列、信号量。该命令运行在 Tuxedo 服务器异常中止情况下。在正常情况下， Tuxedo 应用关闭时自动释放 IPC 资源。清除的 IPC 资源包括 Tuxedo ATMI 服务与客户端的 IPC 资源。

tmipcrm 只能清除本机 IPC 资源，不能用于清除在配置文件中远程机器的 IPC 资源。

使用 tmipcrm 必须指定 TUXCONFIG 文件，TUXCONFIG 必须存在，而且可读。

只有管理员或具有相应权限的人可以运行该命令。

常用参数见表 17-4。

表 17-4

选项	描述
-y	确定执行
-n	不删除 IPC 资源，只输出 IPC 资源列表

tmipcrm 输出实例如下。

示例 17-5：

```
$ tmipcrm /home/user/apps/tuxconfig
Looking for IPC resources in TUXCONFIG file /home/user/apps/tuxconfig
The following IPC resources were found:
Message Queues:
0x2345
0x3456

Semaphores:
0x34567
0x45678

Shared Memory:
0x45678
0x56789
```

```
Remove these IPC resources (y/n)?: y
Removing IPC resources  done!
```

17.3.4 IPC 清除脚本

另一种是使用 ipcrm 开发自定义脚本删除所有的目前正在由这个 UNIX 用户使用的 IPC 资源。这种风格的脚本可以按定制删除用户下所有 IPC 资源，而不是针对特定的 Tuxedo 域中的资源。

为避免误删其他产品或应用使用的 IPC 资源，建议不同的应用程序在不同用户账户下运行。

非 Linux 下脚本运行环境如下。

示例 17-6：

```
ipcrm 'ipcs |grep <username>|awk '{print "-" $1 " " $2}''
```

Linux 下脚本运行环境如下。

示例 17-7：

```
ipcrm 'ipcs -m|grep <username>|awk '{print "shm" " " $2}''
ipcrm 'ipcs -s|grep <username>|awk '{print "sem" " " $2}''
ipcrm 'ipcs -q|grep <username>|awk '{print "msg" " " $2}''
```

17.3.5 bbsread

tmadmin 的 bbsread 子命令可以查看当前 Tuxedo 使用的 IPC 资源状态。

bbsread 输出如下。

示例 17-8：

```
SITE1> bbsread
IPC resources for the bulletin board on machine SITE1:
SHARED MEMORY:          Key: 0x1013c38
SEGMENT 0:
                          ID: 15730
                        Size: 36924
        Attached processes: 12
     Last attach/detach by: 4181

This semaphore is the system semaphore
SEMAPHORE:               Key: 0x1013c38
                          Id: 15666
       |  semaphore  |  current  |   last    | # waiting |
       |   number    |  status   | accesser  | processes |
       |-----------------------------------------------|
```

```
        |    0       |  free   |  4181   |    0     |
        |------------------------------------------|
This semaphore set is part of the user-level semaphore
SEMAPHORE:                Key: IPC PRIVATE
                         Id: 11572
        | semaphore | current | last    |# waiting|
        |  number   | status  |accesser |processes|
        |------------------------------------------|
        |     0     | locked  |  4181   |    0     |
        |     1     | locked  |  4181   |    0     |
        |     2     | locked  |  4181   |    0     |
        |     3     | locked  |  4181   |    0     |
        |     4     | locked  |  4181   |    0     |
        |     5     | locked  |  4181   |    0     |
        |     6     | locked  |  4181   |    0     |
        |     7     | locked  |  4181   |    0     |
        |     8     | locked  |  4181   |    0     |
        |     9     | locked  |  4181   |    0     |
        |    10     | locked  |  4181   |    0     |
        |    11     | locked  |  4181   |    0     |
        |    12     | locked  |  4181   |    0     |
        |    13     | locked  |  4181   |    0     |
        |-----------|---------|---------|----------|
```

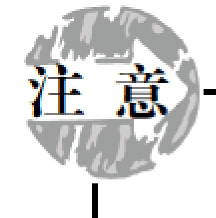
bbsread 输出分为两部分：共享内存与信号量。

其中，共享内存显示的信息的常用参数见表 17-5。

表 17-5

字段	描述
ID	IPC 资源唯一标识
Size	共享内存大小
Attached processes	显示共享内存段中活动进程数。这些进程要处理如 BBL、WSH 进程、Tuxedo 服务器等
Last attach/detach by	最后一个附加到或断开共享内存段的进程的 PID

bbsread 显示所有的信号量是否被锁定，访问它的进程 ID，以及等待的进程数。



ipcs 输出的是信号集的信息，bbsread 输出的是信号集中所有信号量的信息。

17.4　IPC 常见疑难问题

当应用服务器关闭失败时，如何清理 IPC 资源？

当 Tuxedo 的应用程序用 tmshutdown 命令关闭时，所有的 IPC 资源会被 Tuxedo 从系统中自动清除。但在某些情况下，应用程序可能无法正常关闭，这时 IPC 资源继续保留在系统中无法释放。如果发生这种情况，可能无法重新启动应用程序。

一种解决办法是利用脚本文件来调用系统 IPC 命令删除用户下所有的 IPC 资源。然而，使用这个方法很难区分不同的 IPC 资源集合；一部分可能属于 Tuxedo 系统资源，一部分可能属于 Tuxedo 的应用程序；还有一些属于其他应用程序。如果不小心将 IPC 资源错误地删除，对应用程序会造成严重的危害。

使用 Tuxedo 的 IPC 工具（即 tmipcrm 命令）可以清除某一应用范围内活动 IPC 资源，这样可以避免误删其他 IPC 资源。

第 18 章　一般网络故障

Tuxedo 应用网路通信常见的问题包括错误的端口配置、无效的远程或者是本地服务配置、无效的动态 ip 地址和动态端口号以及不正确的防火墙配置。

下面介绍如何解决这些通用的网络配置问题，同时讲解一些诸如防火墙、包跟踪工具、网络状态工具等的使用。

18.1　防火墙及防火墙相关故障

防火墙是一个硬件或者是软件，在互联的网络中，能够阻止被安全策略禁止的信息交互。它的基本功能是在不同可信度的区域之间进行流量控制。典型的可信度区域有 internet（没有可信度的区域）和内部网络（高可信度的区域）。防火墙的目的是通过安全策略和链接模式控制不同可信度区域之间的连接。

合理的防火墙配置是需要技术和智慧的，要求对网络协议和电脑安全有很深的了解。一个很小的错误就能够使防火墙瘫痪，同时也可能影响到 Tuxedo 网络应用和域应用的功能实现。

由于防火墙可以阻止一台机器通过网络与另一台机器的连接，如果 Tuxedo 应用程序不能正常工作，要考虑可能是因为防火墙阻止了 Tuxedo 之间的通信造成的。

18.2　网络状态查询 netstat

netstat 是一个命令行工具，能够显示当前网络输入输出链接的活动状态，netstat 可以在 UNIX，类 UNIX 系统和 Windows 操作系统上使用。

netstat 除查看一些基本网络状态以外，还可以判定一个网络链接是否已经被建立在一台机器的特定端口上。它是检测 Tuxedo 不同组件之间通过相应端口通信的很有用的工具。

例如，在域配置文件的 DM_TDOMAIN 部分，需要指定与本地和远程域所侦听的网络地址，这些配置通过机器名和端口号来指定。为本地域指定的端口号应出现在被 netstat 命令行产生的清单中。如果没有，那么这个域可能没有被正确地配置，也可能没有被成功启动。

对 netstat 传入的参数和通过 netstat 产生的输出，是因机器不同而不同的，甚至在相同的平台上不同的实现也会有不同的参数。在平台上使用 netstat 前，应该查阅当前版本的说明文档。如下是针对 netstat 的 Windows XP 版本的一些合法信息。

当输入 netstat 时将会显示一个特别长的信息列表，所以要掌握一定的方法用来知道所需要的信息，以及怎样显示所需要的信息。

例如，如果仅想要看 TCP 链接信息，那么就使用“netstat –p tcp”，这将会显示一个出入电脑的 tcp 链接列表。

下面的例子显示了在端口 1620、1645、1646、3150、和 7001 连入电脑的 tcp 链接信息。

示例 18-1：

```
C:\temp\tuxs11>netstat -p TCP
Active Connections

Proto  Local Address    Foreign Address                   State
TCP    LANDINGBJ:1620   baym-cs251.msgr.hotmail.com:1863  ESTABLISHED
TCP    LANDINGBJ:1645   207.68.178.61:http                CLOSE WAIT
TCP    LANDINGBJ:1646   210.8.175.254:http                CLOSE WAIT
TCP    LANDINGBJ:3150   mail.lawnscapes.biz:pop3          TIME WAIT
TCP    LANDINGBJ:7001   BUBBLES:0                         LISTENING
```

下面的例子是使用了“netstat -r” 来显示路由表信息。

对于大部分人来说，该命令将会显示一个 ip 和一个网关地址，如果有多个接口，或者一个接口上有多个 ip，在这种情况下该命令可以检测网络路由问题。

示例 18-2：

```
C:\temp\tuxs11>netstat -r

Route Table
======================================================================
Interface List
0x1 ....................MS TCP Loopback interface
0x2 ...00 06 25 43 1e 49 ...... Wireless-G Notebook Adapter - Packet Scheduler Miniport
======================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0      192.168.1.1   192.168.1.103     25
        127.0.0.0        255.0.0.0        127.0.0.1       127.0.0.1      1
      192.168.1.0    255.255.255.0    192.168.1.103   192.168.1.103     25
    192.168.1.103  255.255.255.255        127.0.0.1       127.0.0.1     25
    192.168.1.255  255.255.255.255    192.168.1.103   192.168.1.103     25
        224.0.0.0        240.0.0.0    192.168.1.103   192.168.1.103     25
  255.255.255.255  255.255.255.255    192.168.1.103   192.168.1.103      1
Default Gateway:      192.168.1.1
======================================================================
Persistent Routes:
  None
```

18.3 网络报文追踪

在市场有许多商业化的工具用来分析网络状态，进行包跟踪等。这些工具能用来追踪

从源到目的端的路由。

下边列出了一些常用的工具：

表 18-1

操作系统	工具
Solaris	snoop： snoop 从网络上抓取包，并且将其内容进行显示。snoop 通过使用网络包过滤和流缓存模式来实现高效的包抓取，抓取到的包可以在被接收时显示出来，也能被保存到一个文件中为以后使用 snoop 能够以单行表单和多行表单的形式显示。在一个单行的概括表单中，仅从属于最高协议级别的数据才被显示。例如，一个 nfs 包只用来显示 NFS 信息；底层的 RPC、UDP、IP 和 ehternet 框架信息将不被显示，但是如果选择多行表单时它们能被显示出来
HP	nettl： nettl 用于监测网络事件，如状态改变、出错、链接建立等，也可以抓包 抓包的输出用 netfmt 查看
Linux	tcpdump： tcpdump 打印出匹配了逻辑表达式的接口上包的头信息。它也能以“-w”标识运行，而这个运行模式将保存包数据到一个文件中为日后分析所用，还有一种是“-r”模式，它能够从一个已保存的文件读取包信息，而不是从网络接口中。仅有匹配定义的表达式的包将被 tcpdump 处理 tcpdump 如果不以“-c”运行的话，它将一直抓取包直到它被 SIGINT 信号（通常为输入了中断字符，如 ctrl+c）或者 SIGTERM 信号中断 当 tcpdump 完成包抓取后，它将会报告记录总数： ❑ 通过过滤器接收的包 ❑ 被内核丢掉的包（这种被丢弃的包是由于缺乏缓存空间）
Windows & UNIX (including Linux)	ethereal： ethereal 是一个 UNIX 和 Windows 平台上免费的网络协议分析工具 可以交互式地浏览抓取的数据，查看每个包的概要或详细信息

当确定网络没有把数据从源传递到目的端，但是又不知道什么原因导致，此时跟踪工具将发挥它的作用。数据包进入以太网但不知道在什么地方丢失了，所以希望找到数据包在什么地方丢失。

traceroute（或者在 Windows 机器上的 tracert）顾名思义，按顺序列出了数据包从源到目的地所经过的机器。通过如下 traceroute 列出的机器名和 IP 地址，可以判断出数据包丢失的点。

示例 18-3：

```
C:\temp\simpapp>tracert www.microsoft.com -w 30000

Tracing route to www.microsoft.com.nsatc.net [207.46.198.30]
over a maximum of 30 hops:
```

```
      1    1 ms    1 ms    2 ms  192.168.1.1
      2   32 ms   55 ms   35 ms  nexthop.qld.iinet.net.au [203.55.228.88]
      3   34 ms   31 ms   33 ms  Gi1-0-0-17.bne-pipe-bdr2.chime.net.au
[203.55.228.189]
      4   33 ms   32 ms   33 ms  GigabitEthernet2-1.woo2.Brisbane.telstra.net
[139.130.237.29]
      5   32 ms   35 ms   31 ms  GigabitEthernet6-0.woo-core1.Brisbane.telstra.net
[203.50.51.129]
      6   44 ms   45 ms   45 ms  Pos5-0.ken-core4.Sydney.telstra.net [203.50.6.221]
      7   44 ms   45 ms   47 ms  10GigabitEthernet3-0.pad-core4.Sydney.telstra.net
[203.50.6.86]
      8   47 ms   46 ms   45 ms  10GigabitEthernet2-2.syd-core02.Sydney.net.
reach.com [203.50.13.42]
      9  194 ms  191 ms  199 ms  i-1-0.sjc-core01.net.reach.com [202.84.249.82]
     10  306 ms  200 ms  451 ms  202.84.251.90
     11  213 ms  224 ms  230 ms  134.159.62.38
     12  299 ms  205 ms  207 ms  207.46.43.1
     13  230 ms  232 ms  231 ms  pos4-1.tuk-76cb-1a.ntwk.msn.net [207.46.34.162]
     14  234 ms  233 ms  230 ms  pos1-0.iuskixcpxc1202.ntwk.msn.net [207.46.36.
146]
     15  215 ms  229 ms  230 ms  pos1-0.tke-12ix-1b.ntwk.msn.net [207.46.155.5]
     16  233 ms  232 ms  230 ms  po11.tuk-65ns-mcs-1a.ntwk.msn.net [207.46.224.216]
     17  233 ms  232 ms  230 ms  www.microsoft.com [207.46.198.30]
```

18.4　其他网络工具

18.4.1　ping 命令

ping 可以判断出一台主机是否正常响应，是否可以从测试机通过网络链接到目标主机。它通过传递 ICMP 包到目标主机并等待响应的方式工作。

示例 18-4：

```
C:\temp\simpapp>ping www.adelaide.com

Pinging adelaide.com [202.191.96.222] with 32 bytes of data:

Reply from 202.191.96.222: bytes=32 time=76ms TTL=47
Reply from 202.191.96.222: bytes=32 time=76ms TTL=47
Reply from 202.191.96.222: bytes=32 time=75ms TTL=47
Reply from 202.191.96.222: bytes=32 time=74ms TTL=47

Ping statistics for 202.191.96.222:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
```

```
    Approximate round trip times in milli-seconds:
        Minimum = 74ms, Maximum = 76ms, Average = 75ms
```

从上面的例子中可以看出通过用户的机器可以连接到 www.adelaide.com 这台主机上。

示例 18-5：

```
C:\temp\simpapp>ping www.microsoft.com

Pinging www.microsoft.com.nsatc.net [207.46.199.120] with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 207.46.199.120:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

上面的例子说明从用户的机器上不能 ping 到 www.microsoft.com 这台主机上。

由于内部网络可能受到 ping 命令的影响，所以防火墙管理员经常会在网络中屏蔽掉 ping 命令以限制它对网络的影响。不能 ping 同一台机器时，还需要进一步分析是否是防火墙的原因造成的。

18.4.2　telnet 命令

telnet 是绑定到大多数 Windows 和 UNIX 操作系统中的一条命令，它主要是用来和远端机器建立连接。在一般情况下，telnet 到一台机器上，它会用那台机器的默认端口 23 建立连接。然而，telnet 也可以与指定的任何一个可用的端口会话，在测试连接时不光要指定机器名（IP 地址），还要指定那台机器上的一个可用的端口。

例如，有一个 Tuxedo 域，配置了一个网关 NWADDR bubbles:7010。如果 telnet 到那台机器上并指定正确的端口就不会得到错误信息。然而，如果连接一个不可用的端口，会得到如下错误信息。

示例 18-6：

```
C:\temp\simpapp>telnet landingbj 7011
Connecting To landingbj...Could not open connection to the host, on port
7011: Connect failed
```

当一个 telnet 连接成功并且不是连接到 telnet 服务器上，通常会得到一个空白的窗口。

例如，运行了一个 telnet 到 HTTP 服务器上，会得到一个可以在里面写入命令的空白窗口。事实上如果懂得连接到 HTTP 服务器上连接字符串的格式，就可以在此和 HTTP 服务器进行交互。如果懂得站点结构，还可以在此向 HTTP 服务器提出请求并得到页面。

18.4.3 ifconfig

ifconfig 被用作设置和显示 UNIX 机器上的网络配置信息。它可以被用来分配一台机器的 IP，配置网络子网掩码等信息。

当探查到一个网络错误，出现问题的那台机器的 IP 地址是很重要的，它可以通过 ifconfig 命令找到。

另外，它还显示出了子网掩码和其他一些参数信息让用户知道自己在网络中的位置。

示例 18-7：

```
# ifconfig
eth0      Link encap 10Mbps Ethernet  HWaddr 00:00:C0:90:B3:42
         inet addr 172.16.1.2 Bcast 172.16.1.255 Mask 255.255.255.0
         UP BROADCAST RUNNING  MTU 1500  Metric 0
         RX packets 3136 errors 217 dropped 7 overrun 26
         TX packets 1752 errors 25 dropped 0 overrun 0

lo        Link encap:Local Loopback
         inet addr:127.0.0.1  Mask:255.0.0.0
         UP LOOPBACK RUNNING  MTU:3924  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
         Collisions:0
```

18.4.4 ipconfig

ipconfig 是 Windows 平台下一个非常有用的工具，尤其是可以帮助用户探查网络错误。ipconfig 可以被用来显示 TCP/IP 信息，包括用户的 IP 地址，DNS 服务器地址，网络适配器类型，等等。

如果用 DHCP 动态分配 IP 地址，那么这个工具就非常有用。因为 TCP/IP 信息包括地址是在计算机启动时才分配的，所以直到启动时才会知道。ipconfig 将会在任何时候显示该信息。

示例 18-8：

```
C:\bea\Applications\tuxedo8.1\testapp>ipconfig

Windows IP Configuration


Ethernet adapter Wireless Network Connection 2:

       Connection-specific DNS Suffix  . : iinet.com.au
```

```
        IP Address. . . . . . . . . . . . : 192.168.1.119
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.1.1
```

18.4.5　网络压缩

当大的信息包通过网络传递时，会增加系统的负载。Tuxedo 允许用户将数据压缩，从一个应用程序进程传递到另一个。数据压缩在大多数应用中都非常重要，尤其在大配置的系统中至关重要。在一台机器上发送和接收数据时可以用数据压缩，或者在不同机器间传递也可以使用数据压缩。两种传递方式都有它的优势。

（1）在单机上传递。由于传递通过 IPC 资源，数据压缩可以降低 IPC 资源的使用。

（2）在多机之间传递。数据通过网络传递，数据压缩可以降低网络负载。

是否使用数据压缩取决于所传递的信息包的大小，它可以通过 UBB 配置文件在*MACHINES 段配置参数 CMPLIMIT 作为压缩阀值，超过这个值的数据在发送前将被压缩。

18.5　Tuxedo 多机架构（MP）

18.5.1　Tuxedo MP 应用的注意事项

1. 为域下的每个机器都开启 tlisten 进程

tlisten 进程必须运行在所有主机及从机上。

虽然主机上启动 tlisten 进程不是必须的，但是建议开启。当主机宕机，从机充当主机时，需要原主机上启动 tlisten 进程以使现在的主机能够重新启动在它上面运行的 SERVER。

当应用程序在主机上启动时，它会联系每台从机上的 tlisten 进程确保从机上的 Tuxedo 应用系统启动起来。

该配置是通过 UBB 配置文件的*NETWORK 段下的 NLSADDR 参数指定，经常被指定为//hostname:port_number 形式。

2. 一台机器到另一台机器的 BRIDGE 通信地址

该参数设置在 UBB 配置文件的*NETWEORK 段下，参数为 NADDR，经常被指定为//hostname:port_number 形式。这是 BRIDGE 进程用来监听内部机器之间通信的地址。

18.5.2　负载均衡网络应用程序

在 MP 配置中，一个 SERVICE 可能被配置到一个域下的很多台机器上，当一个客户端连接到应用程序时，它会连接到一个特定的机器（通过本地客户端或者远程客户端），当客户端请求一个 SERVICE 时，Tuxedo 需要决定哪台机器提供这个 SERVICE。

如果负载均衡被开启（在应用程序配置文件的*RESOURCES 段下 LDBAL 被设置成 Y），Tuxedo 将会尝试把请求通过网络平均分配。因为负载信息不会全局更新，所以每台机器储存它自己的负载信息。

在配置文件中的*MACHINES 段的 NETLOAD 参数，可以用来强制地把更多的请求发送到本地队列中。该参数指定了一个加到远端机器的负载数，增大 NETLOAD 的值，会强制 Tuxedo 把更多的请求发送到本地队列中而不是远端机器。

18.5.3 常见问题

如果任何一台域下的机器不能和其他机器通信，则应用程序的部分功能将不能被使用。以下是一些引起通信问题的常见错误。

1. 防火墙错误

如果从机应用系统不能启动，则可能是因为防火墙限制了与 tlisten 进程运行的端口的连接，或者是 tlisten 进程没有运行起来。

如果应用系统启动起来，但是 SERVICE 不可用，很可能是因为防火墙阻止了和 NADDR 端口的通信。

如果网络地址转换（NAT）被用在网络环境中，那么需要在配置 UBBCONFIG 时对 WSL 指定-H 参数。

防火墙会断开那些很长时间不活动的 TCP 连接。如果这种情况出现在用户的环境里，用户可以通过在 domain 环境中设置 keepalive 参数或者在 TCP/IP 协议层设置 keepalive 来保持活动的连接阻止防火关闭连接。

2. 动态 IP 地址问题

一些 Windows 机器在连接到网络时会动态分配 IP 地址，这样在配置文件中很难指定 NLSADDR 和 NADDR 的地址。

每当有一个新的 IP 分配到一台机器上，都要更新 UBB 配置文件中的 IP 地址。

3. 端口号冲突问题

Tuxedo MP 应用系统可能用到很多端口。除了在 UBBCONFIG 配置文件中的*NETWORK 段下指定的 tlisten 和 BRIDGE 的端口，也可能需要给 Workstation 监听和管理进程分配端口号。还可能在这台机器上有其他应用，如 WebLogic 应用或者其他 Tuxedo 应用、后台守护进程、邮件服务进程要使用端口号。

总之，分配给 Tuxedo 应用的端口号必须唯一，这点很重要。

注意有些端口是动态分配的，如 WSH 进程端口，因此在诊断网络连接错误时这一点一定要考虑进去。

当一个远程客户端连接到应用程序时，首先和 WSL 监听进程建立连接，这个进程会监听特定机器的特定端口。一旦建立连接客户端和 WSH 进程就会在另一个端口进行通信。这里的问题在于当 WSH 进程开始与远程客户端用一个特定的端口建立连接时，防火墙管

理员经常发现 WSH 连接是有隐患的，因为它打开了大于 2048 的对外连接端口。在 Tuxedo6.5 及更高版本，有两个新的参数“-p”和“-P”被加到 WSL 命令行，它们提供了 WSH 进程可以使用的对外连接端口的限制，“-P”指定最大值，“-p”指定最小值。这些端口由 Tuxedo 管理员配置，防火墙管理员配合打开这些端口用于和外部网络连接。

最后注意一下 WSL 的“-H”选项。“-H”选项使用一个参数指定外部网络地址。该参数指定了一个完整的网络地址被用作 WSH 进程的地址模板。该地址将结合一个 WSH 网络地址生成一个通过远程客户端连接到 WSH 进程的网络地址。

注意在 Workstation 远程客户端，有两个环境变量可以用来指定在远端机器上的一系列应用程序端口号与 Tuxedo 域建立连接。

相关变量见表 18-2。

表 18-2

环境变量	描述
WSFADDR	远程客户端用于连接到 WSL 或 WSH 时绑定地址的可选范围的起始值。它和 WSFRANGE 变量共同决定了远程客户端可绑定的地址范围
WSFRANGE	远程客户端用于连接到 WSL 或 WSH 时绑定地址的端口范围。它和 WSFADDR 变量共同决定了远程客户端可绑定的本地地址范围

4．无效模式

Tuxedo MP 应用需要把*RESOURCES 节的 MODEL 参数设置成 MP。

18.5.4　用 tmadmin 监控

监控所有的 Tuxedo 应用程序、MP 应用，都可以用 tmadmin 来操作。

下面指出 tmadmin 工具在 Tuxedo MP 模式中的运用。

1．设置监控节点

刚开始进入 tmadmin 输入时，可能会发现很多通常在 SHM 配置中可以显示的信息，在 MP 模式下不显示，并且用“-”代替。

示例 18-9：

```
C:\bea\Applications\tuxedo8.1\testapp>tmadmin
tmadmin - Copyright (c) 1996-1999 BEA Systems, Inc.
Portions * Copyright 1986-1997 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
Tuxedo is a registered trademark.

> psr
Prog Name      Queue Name  Grp Name    ID RqDone Load Done Current Service
---------      ----------  --------    -- ------ --------- ------------
```

```
WSL.exe          00001.00100  GROUP1        100     -        - ( - )
BBL.exe          30003.00000  simon         0       -        - ( - )
BBL.exe          30002.00000  simple        0       -        - ( - )
DBBL.exe         34223        simple        0      109    5450 ..MASTERBB
onesecond.exe    00001.00001  GROUP1        1       -        - ( - )
BRIDGE.exe       558511       simon         1       -        - ( - )
BRIDGE.exe       296367       simple        1       -        - ( - )
twosecond.exe    00001.00010  GROUP1        10      -        - ( - )
twosecond.exe    00001.00011  GROUP1        11      -        - ( - )
fivesecond.exe  fivesec       GROUP2        20      -        - ( - )
fivesecond.exe  fivesec       GROUP2        21      -        - ( - )
fivesecond.exe  fivesec       GROUP2        22      -        - ( - )
fivesecond.exe  fivesec       GROUP2        23      -        - ( - )
fivesecond.exe  fivesec       GROUP2        24      -        - ( - )
```

因为 tmadmin 没有设置成显示特殊机器信息，有些元素没有显示。可以使用“default -m”命令，使 tmadmin 显示特定机器的统计信息。

示例 18-10：

```
> default -m simon
simon> psc
Service Name Routine Name Prog Name  Grp Name  ID    Machine  # Done Status
------------ ------------ ---------  --------  --    -------  ------ ------
FIVESECOND   FIVESECOND   fivesecon+ GROUP2    20     simon        0 AVAIL
FIVESECOND   FIVESECOND   fivesecon+ GROUP2    21     simon        0 AVAIL
FIVESECOND   FIVESECOND   fivesecon+ GROUP2    22     simon        0 AVAIL
FIVESECOND   FIVESECOND   fivesecon+ GROUP2    23     simon        0 AVAIL
FIVESECOND   FIVESECOND   fivesecon+ GROUP2    24     simon        0 AVAIL
```

用“default -m all”来显示所有机器的统计信息。

示例 18-11：

```
simon> default -m all
all> psc
Totals for all machines:
Service Name Routine Name Prog Name  Grp Name  ID    Machine  # Done Status
------------ ------------ ---------  --------  --    -------  ------ ------
ONESECOND    ONESECOND    onesecond+ GROUP1     1    simple        0 AVAIL
TWOSECOND    TWOSECOND    twosecond+ GROUP1    10    simple        0 AVAIL
TWOSECOND    TWOSECOND    twosecond+ GROUP1    11    simple        0 AVAIL
FIVESECOND   FIVESECOND   fivesecon+ GROUP2    20     simon        0 AVAIL
FIVESECOND   FIVESECOND   fivesecon+ GROUP2    21     simon        0 AVAIL
FIVESECOND   FIVESECOND   fivesecon+ GROUP2    22     simon        0 AVAIL
FIVESECOND   FIVESECOND   fivesecon+ GROUP2    23     simon        0 AVAIL
FIVESECOND   FIVESECOND   fivesecon+ GROUP2    24     simon        0 AVAIL
```

示例 18-12：

```
all> psr
Totals for all machines:
Prog Name       Queue Name  Grp Name      ID RqDone Load Done Machine
---------       ----------  --------      -- ------ --------- -------
WSL.exe        00001.00100 GROUP1      100      0          0 simple
BBL.exe        30003.00000 simon         0     54       2700 simon
BBL.exe        30002.00000 simple        0     47       2350 simple
DBBL.exe       34223        simple       0    524      26200 simple
onesecond.exe  00001.00001 GROUP1        1      0          0 simple
BRIDGE.exe     558511       simon        1      0          0 simon
BRIDGE.exe     296367       simple       1      0          0 simple
twosecond.exe  00001.00010 GROUP1       10      0          0 simple
twosecond.exe  00001.00011 GROUP1       11      0          0 simple
fivesecond.exe fivesec     GROUP2       20      0          0 simon
fivesecond.exe fivesec     GROUP2       21      0          0 simon
fivesecond.exe fivesec     GROUP2       22      0          0 simon
fivesecond.exe fivesec     GROUP2       23      0          0 simon
```

从 tmadmin 的输出可以看出以下信息。

首先，machine 列有不止一个机器名，说明是 MP 模式。上面的机器有两个：simple 和 simon。

其次，注意这里总共有两个 BBL 进程在运行。为每个运行的机器配置了一个 BBL，而只在配置中的主机器上一个 DBBL 在运行，这个进程的作用就是确定配置中的机器是不是都在运行。

同时，有两个 BRIDGE 进程在运行，它们是配置中机器的通信桥梁，在配置中的每个机器都将开启一个 BRIDGE 进程。

2．重新连接

当网络失败时会导致 MP 应用中的机器被隔离。Reconnect 命令用来强行连接到一个由于网络问题而失去连接的机器。

例如：

示例 18-13：

```
> reconnect simple simon
simple connected to simon
```

18.6　Tuxedo 的多域架构（Domain）

多域结构允许在维护单独管理控制的域时，不影响其他域的运作。

随着业务的增长，开发的系统可能会增加。这些业务系统可能需要单独的管理，但是同时又需要灵活地跟其他系统进行服务的共享。

当一些域跟另外的域相连时，每一个域都会制定它们要从本地导出的服务和从其他域导入的服务。

每一个多域结构中的域都是 Tuxedo 的一个应用。这些域可能既有 SHM 的配置也有 MP 的配置，这在其他域看来没有什么不同。有关域的配置请参考 8.3 节。

18.6.1 DMCONFIG 常见配置问题

作为单域配置附带而来的网络配置问题，有很多常见的多域配置问题。

1. 端口配置

当配置多域时，在域配置文件 DMCONFIG 中的*DM_TDOMAIN 节，需要为每一个接收和发送通信的域指定网络地址。这里为本地域指定了接收数据的端口，并为所有远程域指定了本地域连接的端口。

所以这里指定的端口必须跟对端域配置文件中指定的端口相同。

2. 本地和远程服务

在域配置文件 DMCONFIG 中，需要指定哪些是本地服务，即导出的服务，哪些是远程服务，即导入的服务。

（1）*DM_EXPORT：通过本地访问点为一个或者多个域导出的本地服务。

（2）*DM_IMPORT：准备从一个或多个域访问点导入的远程服务。

18.6.2 使用 dmadmin 监控 domain

tmadmin 工具用来管理域内的应用，dmadmin 命令用来管理多个域之间的连接。dmadmin 是 DMADM 和 GWADM 服务的一个管理接口。

在默认情况下，dmadmin 在没带参数的情况下会进入管理模式。管理员可以在这个模式下查看或操作域之间的连接。也可以在这个模式下创建、销毁或者重新初始化本地域的 DMTLOG。

在使用“-c”选项，或者调用 config 子命令的情况下，dmadmin 将进入配置模式。应用程序管理员可以利用这个模式来更新或者添加新的配置信息到 BDMCONFIG 文件。

dmadmin 需要使用域管理服务器（DMADM）来管理 BDMCONFIG 文件和网关管理服务器（GWADM）来重新配置域网关组。

1. 连接

示例 18-14：

```
connect (co) -d local_domain_name [-R remote_domain_name]
```

连接本地域网关和远程网关时，如果连接失败而又配置了域网关重新连接就会重复地

自动尝试连接。如果-R 没有指定，那么该命令将被应用到本地网关的所有远程域。

该命令在一个多域环境下域关闭，之后重启，但在没有连接上的情况下非常有用。

2．断开连接

示例 18-15：

```
disconnect (dco) -d local_domain_name [-R remote_domain_name]
```

断开本地域网关和远程网关之间的连接并且不再重连。如果连接是活动的，但是自动连接重试处理发挥作用，那么就关闭自动重新连接功能。如果-R 没有指定，那么该命令会用到本地网关的所有远程域。

3．printdomain

示例 18-16：

```
printdomain (pd) -d local_domain_name
```

该命令用于打印出指定的本地域的信息。这些信息包括一个连接远程域的列表，一个被重试的远程域的列表，被网关进程共享的全局信息以及取决于域类型的额外信息。

当想要从本地域找到的远程 SERVICE 不能被调用到时，此命令非常有效。利用它用户可以查看远程域是否真的连接上了。

4．printstats

示例 18-17：

```
printstats (pstats) -d local_domain_name
```

该命令用于打印出指定的本地域的统计和性能信息。打印出的信息取决于域网关类型。

5．default

示例 18-18：

```
default (d) [-d local_domain_name]
```

该命令用于设定默认的本地域。当用*作为参数时默认值不会被设置。如果命令中没有加入参数，那么当前默认值会被打印出来。

18.7　故障分类排除

18.7.1　Tuxedo MP 应用

如果正在运行一个 MP 的应用，发现服务不可用，可按照下列步骤来解决问题。

（1）检查机器上的服务是不是在运行。如果不是，那么原因可能是该机器 tlisten 进程没有运行。

① 检查 tlisten 进程是否正在运行。

② 如果没有请启动 tlisten。如果 tlisten 无法启动，检查 tlisten 进程是不是在机器上与其他进程的端口相冲突。

③ 检查 NADDR 和 NLSADDR 参数设置是否正确。

④ 如果 tlisten 进程正常运行，参数也配置正确，就尝试启动机器。

⑤ 启动后检查被 NLSADDR 参数指定的端口是否在侦听状态，用 netstat 命令。

（2）检查服务器是否已启动。

（3）排除以上范围内的情况如果仍有故障，则按下列步骤进行检查。

① 用 netstat 命令检查 BRIDGE 的端口（NADDR） 是否在监听可用状态，而不是被其他进程占用。

② 如果防火墙对绑定的端口有要求，则要考虑下面的设置。

表 18-3

环境变量	说明
FADDR	指定由本地机器连接到其他机器上时绑定地址的可选范围的起始值。这个参数和 FRANGE 参数共同决定了可绑定的地址范围 例如，如果 FADDR 地址是//mymachine.bea.com:30000 并且 FRANGE 是 200, 那么所有的本地进程连接到其他机器时，将使用 mymachine.bea.com 机器上 30000 和 30200 之间的端口 如果没有设置，该参数默认为空字符串，这意味着操作系统会随机选择一个本地端口
FRANGE	指定由本地机器连接到其他机器上时绑定地址的端口范围。这个参数和 FRANGE 参数共同决定了可绑定的地址范围

（4）在 UBB 配置中检查指定的模式是否为 MP。

（5）如果计算机使用动态 IP 地址，检查 UBB 代码的配置文件设置是否准确，即使用 ifconfig 或 ipconfig 来确定机器的 IP 地址，并检查那些在配置文件中使用的 IP 地址。

（6）检查数据包是否可以从一台计算机到达目标计算机，用 traceroute（UNIX）或 tracert（Windows）以确定一个数据包可以从一台机器发到目标机器。检查防火墙规则，以确保通信不受阻。

18.7.2　Tuxedo Domain 应用

上述几个步骤，也可以用于诊断一个多域结构中的网络问题。

除了这些，请按照下列步骤排查。

（1）检查本地和远程服务的配置是否正确。

（2）使用 dmadmin 的 printdomain 命令来确定是否真的连接到远程域。

① 如果没有连接，使用 connect 命令连接到远程域。

② 如果连接，尝试使用 disconnect 命令断开，然后再重新连接。

第 19 章　WTC 和 JOLT 支持模式

19.1　重温什么是 WTC 和 JOLT

19.1.1　概述

Tuxedo 提供的 Java 与 Tuxedo 的服务器整合有两种不同的技术，分别为 Java 在线事务（JOLT）和 WebLogic Tuxedo 连接器（WTC）。

两种技术都允许在 Java 内置的系统与运行在 Tuxedo 的系统集成。尽管这两种技术相似，但是它们的实现方式和特性完全不同，并且解决的问题也不同。

19.1.2　WebLogic Tuxedo 连接器介绍

WTC 提供 WebLogic 服务器应用程序与 Tuxedo 服务的互操作能力。连接器允许 Weblgic 服务器客户端调用 Tuxedo 服务，以及 Tuxedo 客户端调用 WebLogic 服务器的 EJB 响应服务请求。

WTC 采用域网关机制使 WebLogic 和 Tuxedo 系统进行通信。在此意义上 WTC 功能是非常类似于两个远程 Tuxedo 域进行通信。

事实上，Tuxedo 角度认为本地 Tuxedo 域在和另一个 Tuxedo 系统进行通信，而不区分是 WebLogic 系统。

19.1.3　JOLT 介绍

JOLT 允许 Java 客户端连接 Tuxedo 服务器。JOLT 系统由许多 Java 类库组成，它与 TuxedoWorkstation Client 库的功能类似，即允许客户端（Java 程序）连接 Tuxedo 远程服务器。

JOLT 中所涉及的其他组件包括 JOLT 监听器（JSL）、Jolt 代理服务器（JSH）以及 jrepository（包含系统服务列表及这些服务的相关配置信息）。

JOLT 还提供了一个称为 JOLT Relay 的组件，它允许 JOLT 传输从同一个网络端口进行转发。以这种方式进行转发通信使得网络能够被更加安全地锁定，因此在配置安全性要求很高或分布式环境时是很有帮助的。

19.2　什么引发 WTC 和 JOLT 故障

19.2.1　JOLT 和 WTC 问题主要的两种形式

JOLT 和 WTC 的问题主要有两种形式：设计与配置。设计问题决定应该使用什么样的技术以及在什么条件下使用。一旦选择了适当的技术，那么就会产生与之相关的一些配置问题。

19.2.2　选择适当技术：JOLT VS WTC

当处理 JOLT 和 WTC 时，最常见的问题就是“我的项目适合使用哪个技术”。这个答案相当简单，可以通过回答下列问题来决定选择哪种技术。

（1）Java 代码是否托管（hosted）在 WebLogic 中。

（2）是否要求双向通信。

（3）在 WebLogic 和 Tuxedo 之间是否需要两阶段提交事务。

JOLT 允许任何 Java 程序（不只是基于 WebLogic 的程序）连接 Tuxedo。

因此，当把一个不是运行在 WebLogic 下的 Java 应用程序连接到 Tuxedo 时，JOLT 是最佳的选择。

WTC 支持从 WebLogic 系统到 Tuxedo 的双向通信。JOLT 不支持双向通信。如果不使用双向通信，而且是 WebLogic 系统与 Tuxedo 服务器进行会话，那么 JOLT 和 WTC 都可以使用。通常在这种情况下建议使用 WTC，JOLT 项目稍微有些复杂和难于管理。

WTC 允许事务在 WebLogic 中发起并传播到 Tuxedo，或者在 Tuxedo 中发起并传播到 WebLogic；而 JOLT 不提供一个跨 WebLogic 和 Tuxedo 事务的整合机制。

19.2.3　引发 JOLT 和 WTC 错误的主要原因

一旦决定使用 JOLT 或 WTC，下一步就是在 Tuxedo 和 WTC 环境下进行 JOLT 和 WTC 的配置，下面将对比配置 JOLT 和 WTC 的不同之处，以及这些系统配置中会遇到的问题。

由于 WTC 和 JOLT 都是基于网络的技术，最常见的问题就是存在于网络和网络配置方面。WebLogic 和 Tuxedo 可能存在于不同区域的网络并且这些区域可能被网络或防火墙所分隔。

当建立连接时确保两种环境之间存在一个真实的网络连接。一种简单方式是通过 telnet 命令来完成操作。简单地使用想连接的那个端口（如想从 WebLogic 连接到 JSL 端口）作为参数，从一台计算机 telnet 登录另一台计算机。

例如，在 WebLogic 上试图连接 Tuxedo（IP 地址为 168.2.32.4）上正在运行的 JSL 的端口，使用下面的命令。

示例 19-1：

```
telnet 168.2.32.4 1020
```

如果远程登录连接成功，将看到一个黑色 telnet 屏幕。如果远程登录未成功，则会显示错误提示信息。在此阶段收到的任何错误都说明是网络阻止两个系统进行通信，需要先解决这些网络问题然后才能继续。

通过网络锁定保护它们免受来自于内部和外部的攻击已经变得越来越普遍了。在这种情况下，往往 JOLT 和 WTC 的系统运行端口需要通过防火墙被显式地打开。

WTC 使用 Tuxedo 的域模型进行通信，因此常常使用单一的端口。JOLT 使用 JSL 方式，它与 Tuxedo 中使用的 WSL 方式类似，除初始化连接时的端口以外可能会占用大量端口。如果在用户的环境中使用 JOLT 需要限制端口的数量，那么应该考虑使用 JOLT relay 来帮助其完成操作。较晚的 JSL 版本也允许更改 JSL 和 JSH 端口的限制范围。

19.3 WTC 和 JOLT 相关故障的症状及解决方法

19.3.1 JOLT 常见问题及解决方法

JOLT 最常见的问题分别出现在 JOLT Repository Editor、JOLT Repository 和 JOLT Relay。

1. JOLT Repository Editor

JOLT Repository Editor 是基于 Java applet 的编辑器，它允许用户配置 Tuxedo 公开的访问服务，并提供这些服务的接口。JOLT Repository 编辑器所基于的小程序是容易受到在客户端已经加载的浏览器的 Java 运行环境版本变化的影响。

大多数 JOLT 高级管理员使用 JOLT Bulk Loader，相比 JOLT Repository Editor，它具有很多的优势（不限于下面提到的这些）。

（1）JOLT 的配置可以进行来源控制并且方便保存。

（2）大容量的导入文件易于定义大批量服务。

（3）它无需使用 JOLT Repository Editor，因此提高了可靠性与环境的安全性。

JOLT Bulk Loader 是做批量处理的，如果需要在用户的环境中广泛使用 JOLT，可以考虑使用这个工具，其详细资料可以在 Oracle 官方网站获取。

2. JOLT Repository

JOLT 使用 JSL 对 JOLT 连接进行监听，再通过使用 JSH 进行进一步的处理操作。除 JSL 之外，JOLT 还要求 JOLT 库的配置。

应用中至少要配置一个 JOLT 库服务器 JREPSVR，然而可以在环境中配置不止一个 JREPSVR 用来分担负载。

如果配置多个 JREPSVR，那么只能有一个 JREPSVR 可以在 CLOPT 上设置“-W”标

志。“-W”标志着该服务器可以进行写操作，因此 JOLT Repository 发生的任何变化都会通过该服务器。“-P”标志指向实际的 JOLT Repository。可以在$TUXDIR/udataobj/jolt 目录下找到一个空白的 jrepository 文件。如果多个 JREPSVR 在多个机器上运行，那么 jrepository 文件应该通过使用共享文件系统（或者类似的方式）在所有机器间共享。

示例 19-2：

```
*GROUPS
JREPGRP        GRPNO=94 LMID=SITE1
*SERVERS
JREPSVR SRVGRP=JREPGRP SRVID=98
RESTART=Y GRACE=0 CLOPT="-A -- -W -P /app/jrepository"
JREPSVR SRVGRP=JREPGRP SRVID=97
RESTART=Y RQADDR=JREPQ GRACE=0 CLOPT="-A -- -P /app/jrepository"
JREPSVR SRVGRP=JREPGRP SRVID=96
RESTART=Y RQADDR=JREPQ REPLYQ=Y GRACE=0 CLOPT="-A -- -P /app/jrepository"
```

当配置 JREPSVR 时确保下面信息。

（1）确保只有一个 JREPSVR 在整个域中设置“-W”参数。

（2）运行足够 JPEPSVR 以确保这些服务器中没有排队现象发生。

（3）始终使用$TUXDIR/udataobj/Jolt 下的 jrepository 作为 jrepository 的基础，在此之上添加服务，删除不需要的服务。

不要手动修改 jrepository 文件，否则文件更改后可能无法使用 Repository Editor。

可能发生在 JREPSVR 的一个问题出在 jrepository 文件的缓存区域。如果 jrepository 文件经常变化，可能导致 JREPSVR 缓存区与 jrepository 文件不同步。如果发生这样的情况，则需要在 JREPSVR 中调用隐藏的服务命令“FLUSHCACHE”来刷新缓存区。但是如果运行不止一个 JREPSVR，则无法显式地调用每个服务器上的“FLUSHCACHE”服务。在这种情况下最好关闭其他的 JREPSVR 实例，直到只有一个实例在运行时再调用“FLUSHCACHE”，最后重启刚才关闭的 JREPSVR 的实例。这样可以确保 JREPSVR 缓存区被清除了。

在使用上述方式时，确保系统处于系统低负载期，因为机器上的 JREPSVR 减少，性能会降低。

3. JOLT Relay

JOLT Relay 组件允许 JOLT 流量通过单一网络端口行转发。由 JOLT Relay 把流量传递给 JSL。在处理互联网客户端运行的 appletJOLT 的系统时，Relay 就显得特别有用。同样 JOLT 客户端在 WebLogic 内使用而 WebLogic 环境被防火墙分隔成不同的网络区域时，Relay 也很有用。在这种情况下，JOLT Relay 由两个组件组成，第一个组件是 JOLT Relay（JRLY），它的职责是从 JOLT 客户端到第二个组件 JOLT Relay Adapter（JRAD）进行转发传输。

配置文件内容如下。

示例 19-3：

```
LOGDIR=<LOG_DIRECTORY_PATH>
ACCESS LOG=<ACCESS FILE NAME in LOGDIR>
ERROR_LOG=<ERROR_FILE_NAME in LOGDIR>
LISTEN=<IP:Port combination where JRLY will accept connections>
CONNECT=<IP:Port combination associated with JRAD>
SOCKETTIMEOUT=<Seconds for socket accept()function>
```

LISTEN 端口表明 JOLT Relay 该监听的端口。这是 JOLT Relay 运行的本地 IP 地址。CONNET 属性允许用户指定通过 JOLT Relay 连接到的 JOLT Relay Adapter 的地址和端口。

检查运行 JRLY 的机器是否能连接到运行 JRAD 的机器是非常重要的。如果不能通过 telnet 连接对应的端口，那么表示 JRLY 不可使用。

JRAD 服务器在 UBBCONFIG 配置文件中作为 Tuxedo 服务器启动。正确的服务器启动参数如下所示。

示例 19-4：

```
JRAD    SRVGRP=JSLGRP   SRVID=60
       CLOPT="-A -- -l //machine1:1000 -c //machine1:1050"
```

这里的“-l”是 JRAD 监听的地址（应该与配置文件中 CONNECT 属性设置的值相同），而“-c”参数则是 JRAD 连接到 JSL 的地址。

19.3.2　WTC 常见问题及解决方法

WTC 配置 WebLogic 和 Tuxedo 通过域网关协议建立连接。域网关是 Tuxedo 的一个概念，它允许独立的 Tuxedo 域之间彼此进行通信。在 WTC 中使用了相同的技术允许 Tuxedo 服务器连接 WebLogic 服务器。

WebLogic 同样也提供在 Tuxedo 和 WebLogic 环境中配置 WTC 的很好的样例，WTC 样例可以在 WebLgoic /samples/server/examples/src/examples/wtc 目录找到。在这个目录中会发现预先配置的 Tuxedo 环境包含了可以让系统运行的 UBBCONFIG 和 DMCONFIG 的配置。

如果想配置一个这样的环境，那么它是一个配置简单 WTC 环境的极好的例子。

当 WebLogic 通过 WTC 与 Tuxedo 相连接时，可以在日志文件中设置跟踪级别，这样 WTC 将会产生大量的跟踪日志。可以在 WebLogic 首次启动时设置 WebLgoic WTC 跟踪级别属性来提高跟踪级别。也可以把如下参数直接传递给 Java 可执行文件或设置环境变量。

示例 19-5：

```
JAVA_OPTIONS=-Dweblogic.wtc.TraceLevel=10000
```

表 19-1 是跟踪级别的应用设置。

表 19-1

数值	跟踪组件	描述
10000	TBRIDGE_IO	Tuxedo 队列桥 IO
15000	TBRIDGE_EX	更多 Tuxedo 队列桥信息
20000	GWT_IO	网关 IO，包括 ATMI
25000	GWT_EX	更多网关信息
50000	JAMTI_IO	JAMTI IO，包括低级别的 JAMTI 调用
55000	JAMTI_EX	更多 JAMTI 信息
60000	CORBA_IO	CORBA IO
65000	CORBA_EX	更多 CORBA 信息
100000	All Components	所有 WTC 的相关信息

为了确保将调试信息写入到 WebLogic 控制台的日志文件中，选择运行 WTC 的服务器，单击服务器的 logging 标签，然后在 gerneral 下选择 Debug level 设置 Stdout 和 Stdout 的阈值为 info。

使用 WTC 最常见的问题莫过于在 WebLogic 和 Tuxedo 之间建立各种连接时网络端口不正确的配置。当使用 WTC 时，要确保 WebLogic 和 Tuxedo 端口已经正确设置。另外确保想发布和调用的服务在对应的 WebLogic 和 Tuxedo 域中已经配置为远程和本地服务。

19.4 WTC 和 JOLT 故障排查清单

19.4.1 WTC 故障排除步骤

（1）通过 WTC 的文档验证 WTC 是否配置正确。

（2）启动 WebLogic 和 Tuxedo 系统，查看两端日志文件中是否有错误。

（3）通过调用 Tuxedo 或 WebLogic 中的远程服务来测试连接，确保遍历整个网络。

（4）如果出现问题，提高 WTC 日志记录级别，并检查日志的详细信息。

（5）请记住 WTC 使用 Tuxedo 域体系结构，如果域出现问题则 WTC 也会出现问题。

（6）使用 WebLogic 已安装的 WTC 样例作为参考。（/bea/weblogic**/samples/server/examples/src/examples/wtc）

（7）获得进一步的帮助，请参阅 Oracle 官方联机文档。（http://download.oracle.com/docs/cd/E15261_01/tuxedo/docs11gr1/index.html）

19.4.2 JOLT 故障排除步骤

（1）当配置 JOLT 系统时确保您有至少一个 JSL 和 JREPSVR 在环境中运行。

（2）确保 jrepository 文件中只有一个 JREPSVR 进程是可写的；

（3）如果 JOLT Repository Editor 造成的问题可以考虑，使用其替代方法 JOLT Bulk Loader。

（4）另外使用更新版本的 Java 运行环境能使 JOLT Repository Editor 更好地工作。

（5）当使用 JRLY 和 JRAD 时，确保所有组件已经运行以及每个组件之间的网络通信能正常工作。考虑安装 JRLY 作为服务或守护进程，以使它一直保持运行。

（6）获得进一步的帮助，请参阅 Oracle 官方联机文档。（http://download.oracle.com/docs/cd /E15261_01 /tuxedo/docs11gr1/index.html）

第 5 篇

高　阶　篇

第 20 章　Tuxedo 的 COBOL 编程

COBOL 是 Common Business Oriented Language（面向商业的通用语言）的缩写，又称为企业管理语言、数据处理语言等。它是最早的高级编程语言之一，是世界上第一个商用语言。COBOL 主要是应用于银行、金融和会计行业等非常重要的商业数据处理领域。

虽然目前更多的新系统使用 Java 和 EJB 技术，但对于已有系统，特别是主机遗留系统，重新改写 COBOL 语言的应用软件也要花上很长的时间，对于关键业务这样做风险很高，因此 COBOL 在一段时间内不会消失。对于主机遗留系统，将 COBOL 程序迁移到开放平台，不改动原来的业务逻辑是降低成本，并使原有系统可以基于开放标准和更多平台系统互联的捷径。

Tuxedo 是开放平台上最好的交易中间件之一，也是支持主机关键业务遗留系统迁移的最好平台之一。

20.1　运行环境配置

Tuxedo 的所有功能，包括请求/应答式通信、会话通信、队列通信、事件代理和消息通知、数据依赖路由、服务优先级等，对 COBOL 应用都支持。而 Tuxedo 运行 COBOL 客户端/服务器的配置、运行和管理与 C 程序的应用没有区别，非常方便。同一套应用中可以既有 C 程序，又有 COBOL 程序，它们之间也可以互相调用。运行 COBOL 客户端/服务器只需要根据不同的 COBOL 编译器的要求，设置相应的环境变量即可。

20.1.1　Tuxedo COBOL 数据记录类型

1．数据记录类型概述

Tuxedo 的 COBOL 应用程序之间互相发送数据时，发送程序首先将发送的数据放到一条记录里面，在 Tuxedo ATMI 用定义好的记录来把消息发送给 ATMI 服务端程序。Tuxedo 的 COBOL 类型记录（Typed Records）具有格式化和自描述的特点，克服了异构平台的差异，为不同系统下的数据表示提供了统一的格式，使 Tuxedo 跨平台进行联机交易、数据转换和数据依赖路由成为可能。

类型记录见表 20-1。

2．定义记录类型

TPTYPE-REC 作为 COBOL 程序中的一个 copybook，用来定义 Tuxedo 类型数据。

下面是 TPTYPE-REC 的各个域和说明（在 copybook 中的形式）。

表 20-1

类型记录	用途	是否自描述	子类型	数据依赖路由	编解码
CARRAY	未定义的字符数组之中，任何一项都可以是 LOW-VALUEA	No	No	No	No
FML	Tuxedo 中私有的一种类型，是一种自描述的类型，这种类型的数据项携带了一个唯一的 ID，可以对其进行数据依赖路由，具有更高的灵活性	Yes	No	Yes	Yes
FML32	与 FML 类型是一样的，支持更多的数据项和数据长度	Yes	No	Yes	Yes
STRING	以 LOW-VALUE 结尾的字符数组，Tuxedo 可以对其在应用不同字符集的机器上实现自动转换	No	No	No	No
VIEW	在应用程序中定义数据结构的类型	No	Yes	Yes	Yes
VIEW32	与 VIEW 一样，支持更多的数据项和数据长度	No	Yes	Yes	Yes
X_COMMON	与 VIEW 类似，用来在 C 和 COBOL 之间转换，数据项的类型只能是 short、long 和 string	No	Yes	Yes	Yes
X_OCTET	与 CARRAY 用法相同	No	No	No	No

（1）05 REC-TYPE PIC X(8)。

（2）88 X-OCTET VALUE “X_OCTET”。

（3）88 X-COMMON VALUE “X_COMMON”。

（4）05 SUB-TYPE PIC X(16) 。

（5）05 LEN PIC S9(9) COMP-5。

（6）88 NO-LENGTH VALUE 0。

（7）05 TPTYPE-STATUS PIC S9(9) COMP-5。

（8）88 TPTYPEOK VALUE 0。

（9）88 TPTRUNCATE VALUE 1。

REC-TYPE：定义用哪种记录类型来接收和发送。

SUB-TYPE：定义记录的子类型。

LEN：数据在发送时，这里的长度就是发送数据所占的字节数，当数据传送完以后 LEN 的值就变成了传送了多少字节的值，当数据在接收时 LEN 标记了要被 move 到数据记录中的字节数；在调用成功后，LEN 存放了实际被 move 到数据记录中的字节数，如果传进来的消息比定义的 LEN 大，这个数据就被截断，超出的部分被丢弃，同时 TPTYPE-STATUS 被设置成 TPTRUNCATE。

3．等效数据类型定义

Tuxedo 支持 COBOL 应用和 C 应用相互调用，并在调用时自动进行数据匹配和转换，相关规则见表 20-2。

表 20-2

C 数据类型	对应的 COBOL 数据类型
Float	COMP-1
Double	COMP-2
Long	S9(9)COMP-5
Short	S9(4)COMP-5
Dec_t	COBOL COMP-3 压缩的十进制数据类型

COMP-5 是 MicroFocus COBOL 支持的类型，用以使 COBOL 的整数与对应的 C 语言数据格式匹配，在 VS COBOL II 中这个类型对应的是 COMP。

为了提高存储效率，COBOL 支持压缩的十进制数据类型（COMP-3），两个十进制类型的数字压缩到一个字节中，在低八位存储数据的符号。这种类型的长度可以有 1～9B，可以存储带符号的 1～17 位长的数字。

Tuxedo VIEW 定义了 dec_t 数据类型，大小用被逗号分开的两个值来表示：第一个值表示 COBOL 中 COMP-3 类型数据所占的字节数，第二个值表示 COBOL 中小数点右边的小数位数。可以用下面的公式做转换。

dec_t(m, n) => S9(2*m-(n+1),n)COMP-3

例如，一个大小被定义为 6,4 的 dec_t 类型数据，表示小数点右边是 4 个数字，左边是 7 个数字，后面的半个字节用来存储符号位，COBOL 程序中它的定义是 S9(7)V9(4).。

注 意

FML 不支持 dec_t 类型，如果使用了 FML VIEW ，那么 VIEW 的每个域都要与 C 语言里面的数据类型匹配。比如，一个压缩的十进制数需要与 FML 的 string 匹配，然后才能进行数据转换。

4．如何使用 VIEW 记录类型

VIEW 定义的记录有两种，第一种是 FML VIEW，是由 FML 产生的一种 COBOL 记录格式，第二种是独立的 COBOL 记录类型。

另外，之所以把 FML 记录转换成 COBOL 格式并且再转换回来供 C 使用，是因为 FML 的函数不支持 COBOL 编程环境。

用 VIEW 定义的数据类型，需要遵守下面的步骤。

（1）设置对应的环境变量。

（2）在 VIEW 描述文件中定义每个结构。

（3）用 Tuxedo 的 VIEW 编译器编译 VIEW 描述文件，命令是：viewc-C。

用这个命令可以产生一个或者多个 copybook，每个 copybook 都包括对变量的数据定义，这些 copybook 可以按照开发者的需求在 LINKAGE 或 WORKING STORAGE 中使用。

给 VIEW 记录类型设置如下环境变量。

（1）FIELDTBLS/FIELDTBLS32。

（2）FLDTBLDIR/FLDTBLDIR32(FML VIEW) 。

（3）VIEWFILES/VIEWFILES32。

（4）VIEWDIR/VIEWDIR32。

1. 创建 VIEW 描述文件

要使用 VIEW 记录类型，首先要在 VIEW 文件中定义 COBOL 记录格式，VIEW 文件中第一行定义 VIEW 名字，下面定义数据项及其属性。

（1）type：要定义域的数据类型，可以是 short、long、float、double、char、string 或 carray。

（2）cname：对应在 COBOL 中引用的变量名。

（3）fbname：如果使用 FML-VIEW 或者 VIEW-FML 转换的话，需要在这里定义跟 FML 相对应的变量名，而且必须跟 FML 文件中定义的一样。

（4）count：这个变量需要被定义的次数，对应 COBOL 中的表。

（5）flag：定义转换规则，可选的值有 P、S、F、N、C、L。

（6）size：为 STRING 和 CARRAY 类型定义最大长度，其他类型忽略此处。

（7）null：变量的初始值。

一个 FML VIEW 如下所示。

示例 20-1：

```
$ /* View structure */
VIEW MYVIEW
#type      cname      fbname     count   flag    size    null
float      float1     FLOAT1     1       -       -       0.0
double     double1    DOUBLE1    1       -       -       0.0
long       long1      LONG1      1       -       -       0
short      short1     SHORT1     1       -       -       0
int        int1       INT1       1       -       -       0
dec t      dec1       DEC1       1       -       9,16    0
char       char1      CHAR1      1       -       -       '\0'
string     string1    STRING1    1       -       20      '\0'
carray     carray1    CARRAY1    2       CL      20      '\0'
END
```

独立的 VIEW 如下所示。

示例 20-2：

```
$ /* View data structure */
VIEW MYVIEW
#type      cname      fbname     count   flag    size    null
float      float1     -          1       -       -       -
double     double1    -          1       -       -       -
long       long1      -          1       -       -       -
short      short1     -          1       -       -       -
int        int1       -          1       -       -       -
dec t      dec1       -          1       -       9,16    -
char       char1      -          1       -       -       -
string     string1    -          1       -       20      -
carray     carray1    -          2       CL      20      -
END
```

独立的 VIEW 与 FML VIEW 的差别主要就在 fbname 与 null 。

2．编译 VIEW

用命令 viewc32-C 来编译 VIEW，后面的参数跟着 VIEW 的名字。这里如果编译的是独立的 VIEW，则需要加上参数-n，另外，还有一个可选的参数-d，用来制定编译结果输出的目录，如编译依赖 FML 的 VIEW 文件 test.v： viewc32 -C test.v。

编译输出的结果如下。

（1）一个或者多个 coopbook，如 TEST.cbl。

（2）一个包含数据结构的头文件，用来供 C 的程序共享使用。

（3）描述文件相对应的二进制文件，如 test.V。

产生的 copybook 的内容如下。

示例 20-3：

```
* VIEWFILE: "test.v"
* VIEWNAME: "TEST"
05 FLOAT1                     USAGE IS COMP-1.
05 DOUBLE1                    USAGE IS COMP-2.
05 LONG1                      PIC S9(9) USAGE IS COMP-5.
05 SHORT1                     PIC S9(4) USAGE IS COMP-5.
05 FILLER                     PIC X(02).
05 INT1                       PIC S9(9) USAGE IS COMP-5.
05 DEC1.
07 DEC-EXP                    PIC S9(4) USAGE IS COMP-5.
07 DEC-POS                    PIC S9(4) USAGE IS COMP-5.
07 DEC-NDGTS                  PIC S9(4) USAGE IS COMP-5.
* DEC-DGTS is the actual packed decimal value
07 DEC-DGTS                   PIC S9(1)V9(16) COMP-3.
07 FILLER                     PIC X(07).
05 CHAR1                      PIC X(01).
05 STRING1                    PIC X(20).
05 FILLER                     PIC X(01).
05 L-CARRAY1 OCCURS 2 TIMES PIC 9(4) USAGE IS COMP-5.
* LENGTH OF CARRAY1
05 C-CARRAY1                  PIC S9(4) USAGE IS COMP-5.
* COUNT OF CARRAY1
05 CARRAY1 OCCURS 2 TIMES     PIC X(20).
05 FILLER                     PIC X(02).
```

要使用该文件必须在程序中用 COPY 语句来引用。在前面的例子中编译器自动加入了 FILLER 这个变量，目的是跟 C 中的数据结构保持一致。

这里的约束是不能从 COBOL 直接将 VIEW 数据传递给 C 的函数，必须通过 ATMI 的这种机制，因为 COBOL 跟 C 的数字表示方法是不同的。

用 viewc 编译产生的头文件可以使用户在 C 与 COBOL 之间互相调用服务和客户端程序。

20.1.2　如何使用 FML 数据类型

FML 接口实际上是为 C 语言设计的，对于 COBOL，可以允许用户把接收到的 FML 记录类型转换成 COBOL 的记录类型，然后进行处理，再把记录转换回 FML 格式的记录类型传回来进行通信。

要使用 FML 记录类型，首先要执行以下步骤。

（1）设置适当的环境变量。

（2）在 FML 域表中描述有关的数据项。

（3）用 FINIT 来初始化 FML 记录。

（4）由 FML 定义生成一个头文件，并且在 C 的语句中使用#include 包含这个头文件。

1．设置环境变量

要在程序中使用 FML 记录类型，首先要对下面环境变量做设置：

FIELDTBLS32：FML32 域对应表的文件名，多个用逗号分隔。

FLDTBLDIR32：FML32 域对应表文件的路径，多个用冒号分隔。

2．创建一个域表文件

使用 FML32 和 FML 依赖的 VIEW 时必须要有对应的域表文件。

下面的格式描述了 FML 域对照表中的每一个域的定义。

示例 20-4：

```
$ /* FML structure */
*base value
name     number  type    flags   comments
```

（1）*base value：为后面域定义了一个开始的偏移量，这样每个域对应的数字就可以复用了，Tuxedo 中的域号 1～100、6000～7000 是给内部使用的。

（2）name：域的标识符，最大长度是 256 长的字符类型，可以用字母和下阵线。

（3）number：域的相对数字值，这个值会和 base 指定的值相加得到实际值，作为域 ID。

（4）type：域的类型，可以是 char、string、short、long、float、double 或者 carray。

（5）flag：保留关键字。

（6）comments：可选注释项。

下面是一个 FML VIEW 的域表例子。

示例 20-5：

```
# name    number    type         flags   comments
FLOAT1    110       float         -        -
DOUBLE1   111       double        -        -
LONG1     112       long          -        -
SHORT1    113       short         -        -
INT1      114       long          -        -
```

```
DEC1     115      string     -      -
CHAR1    116      char        -      -
STRING1  117      string     -      -
CARRAY1  118      carray     -      -
```

3. 初始化一个记录类型

FML 记录类型需要用 FINIT 过程来对其进行初始化。如果 TPNOCHANGE 为真，那么程序接收到的 FML 记录类型（或者刚被程序创建的）将会自动初始化，这样就不需要调用 FINIT。

下面是一个 FML/VIEW 转换的程序。

示例 20-6：

```
WORKING-STORAGE SECTION.
*RECORD TYPE AND LENGTH
01 TPTYPE-REC.
COPY TPTYPE.
*STATUS OF CALL
01 TPSTATUS-REC.
COPY TPSTATUS.
* SERVICE CALL FLAGS/RECORD
01 TPSVCDEF-REC.
COPY TPSVCDEF.
* TPINIT FLAGS/RECORD
01 TPINFDEF-REC.
COPY TPINFDEF.
* FML CALL FLAGS/RECORD
01 FML-REC.
COPY FMLINFO.
*
*
* APPLICATION FML RECORD - ALIGNED
01 MYFML.
05 FBFR-DTA OCCURS 100 TIMES PIC S9(9) USAGE IS COMP-5.
* APPLICATION VIEW RECORD
01 MYVIEW.
COPY MYVIEW.
.....
* MOVE DATA INTO MYVIEW
.....
* INITIALIZE FML RECORD
MOVE LENGTH OF MYFML TO FML-LENGTH.
CALL "FINIT" USING MYFML FML-REC.
IF NOT FOK
MOVE "FINIT Failed" TO LOGMSG-TEXT
```

```
PERFORM DO-USERLOG
PERFORM EXIT-PROGRAM
END-IF.

* Convert VIEW to FML Record
SET FUPDATE TO TRUE.
MOVE "MYVIEW" TO VIEWNAME.
CALL "FVSTOF" USING MYFML MYVIEW FML-REC.
IF NOT FOK
MOVE "FVSTOF Failed" TO LOGMSG-TEXT
PERFORM DO-USERLOG
PERFORM EXIT-PROGRAM
END-IF.

* CALL THE SERVICE USING THE FML RECORD
MOVE "FML" TO REC-TYPE IN TPTYPE-REC.
MOVE SPACES TO SUB-TYPE IN TPTYPE-REC.
MOVE LENGTH OF MYFML TO LEN.
CALL "TPCALL" USING TPSVCDEF-REC
TPTYPE-REC
MYFML
TPTYPE-REC
MYFML
TPSTATUS-REC.
IF NOT TPOK
MOVE "TPCALL MYFML Failed" TO LOGMSG-TEXT
PERFORM DO-USERLOG
PERFORM EXIT-PROGRAM
END-IF.

* CONVERT THE FML RECORD BACK TO MYVIEW
CALL "FVFTOS" USING MYFML MYVIEW FML-REC.
IF NOT FOK
MOVE "FVFTOS Failed" TO LOGMSG-TEXT
PERFORM DO-USERLOG
PERFORM EXIT-PROGRAM
END-IF.
```

在上述程序中 FVSTOF 用来把 VIEW 记录转换成 FML 记录，view 通过 copy 用 view 编译器生成的 copybook 来引入。FML-REC 提供了 VIEWNAME 和 FML-MODE 传输模式，可以设置成 FUPDATE、FOJOIN、FJOIN 或 FCONCAT。

FVFTOS 则用来把 FML 记录转成 VIEW 记录，使用的参数和 FVSTOF 一样，但是不需要设置 FML-MODE 值，系统把与域记录中的值根据 VIEW 中的描述放到数据结构中，如果在 COBOL 的记录中没有为域记录定义的元素，则系统默认把这个元素的值置为 null。

另外，对于 COBOL 中的数组问题，依然采用“超出长度截断，不够长度补空”的原则。

对于 FML32 和 VIEW32，对应的要用 FINIT32、FVSTOF32 及 FVFTOS32，开关 FML-STATUS 取决于操作的结果成功与否，成功则设置成 FOK，否则设置成非零的数值。

4. 创建 FML 头文件

为了能够在客户端程序和服务器子程序中使用 FML 数据类型，必须要创建一个 FML 头文件，并且在应用程序中使用#include 引用它。

使用 mkfldhdr 命令用域表文件创建头文件，例如，创建一个名称为 myview.flds.h 的头文件，用下面的命令。

示例 20-7：

```
mkfldhdr myview.flds
```

当然对于 FML32 来说，要用 mkfldhdr32 来创建。

下面是由 mkfldhdr 生成的一个头文件。

示例 20-8：

```
/* fname fldid */
/* ----- ----- */
#define FLOAT1     ((FLDID)24686)    /* number: 110 type: float */
#define DOUBLE1    ((FLDID)32879)    /* number: 111 type: double */
#define LONG1      ((FLDID)8304)     /* number: 112 type: long */
#define SHORT1     ((FLDID)113)      /* number: 113 type: short */
#define INT1       ((FLDID)8306)     /* number: 114 type: long */
#define DEC1       ((FLDID)41075)    /* number: 115 type: string */
#define CHAR1      ((FLDID)16500)    /* number: 116 type: char */
#define STRING1    ((FLDID)41077)    /* number: 117 type: string */
#define CARRAY1    ((FLDID)49270)    /* number: 118 type: carray */
```

20.1.3 Tuxedo COBOL 客户端编程

1. 注册一个客户端

在一个 ATMI 客户端程序能够执行服务请求之前，首先要在 Tuxedo ATMI 应用中注册该程序，一旦被注册以后就可以开始发送请求和接收应答了。

Tuxedo 的 COBOL 客户端显式注册要通过调用 TPINITAILIZE() 来实现，具体逻辑如下。

示例 20-9：

```
01 TPINFDEF-REC.
COPY TPINFDEF.
01 USER-DATA-REC    PIC X(any-length).
01 TPSTATUS-REC.
COPY TPSTATUS.
CALL "TPINITIALIZE" USING TPINFDEF-REC USER-DATA-REC TPSTATUS-REC.
```

客户端通过发起一个服务请求（或者一个 ATMI 调用）也可以隐式注册，而不是首先调用 TPINITAILIZE()，这时 TPINITAILIZE 是被 Tuxedo 系统用 SPACES 作为参数代替客户端程序自动调用的。

TPINFDEF-REC 是 Tuxedo 系统定义好的一个记录，当客户端程序注册到应用时，通过该记录来传递客户端的 ID，授权信息给系统，并可以指定通知的处理方式，系统访问的方式，以及程序采用单 CONTEXT 连接，还是多 CONTEXT 连接。它也作为一个 COBOL 的 copybook 被引用。

定义格式如下。

示例 20-10：

```
05 USRNAME                    PIC X(30).
05 CLTNAME                    PIC X(30).
05 PASSWD                     PIC X(30).
05 GRPNAME                    PIC X(30).
05 NOTIFICATION-FLAG          PIC S9(9) COMP-5.
88 TPU-SIG VALUE 1.
88 TPU-DIP VALUE 2.
88 TPU-IGN VALUE 3.
05 ACCESS-FLAG                PIC S9(9) COMP-5.
88 TPSA-FASTPATH VALUE 1.
88 TPSA-PROTECTED VALUE 2.
05 CONTEXTS-FLAG              PIC S9(9) COMP-5.
88 TP-SINGLE-CONTEXT VALUE 0.
88 TP-MULTI-CONTEXTS VALUE 1.
05 DATALEN                    PIC S9(9) COMP-5.
```

当客户端程序注册到 Tuxedo 应用时，系统分配给它一个唯一的 ID，这个 ID 传递给它调用的每一个服务，也可以用来作为其他进程给该客户端发通知时的标识。

2．注销一个应用

当所有的服务请求和应答都处理完毕以后客户端程序使用 TPTERM()来注销应用。TPTERM 的结构如下。

示例 20-11：

```
01 TPSTATUS-REC.
COPY TPSTATUS.
CALL "TPTERM" USING TPSTATUS-REC.
```

3．建立一个客户端

用命令 buildclient 来编译出一个可执行的 ATMI 客户端程序。其中程序用的文件都要指定。

命令格式如下。

示例 20-12：

```
buildclient -C filename.cbl -o filename -f filenames -l filenames
```

其中-C 表示要编译的是 COBOL 程序，其他参数和编译 C 程序一样。

4. 一个简单的例子

下面的伪代码为客户端应用从注册应用开始到注销应用的过程。

示例 20-13：

```
⋮
Check level of security
CALL TPSETUNSOL to name your handler routine for TPU-DIP
get USRNAME, CLTNAME
prompt for application PASSWD
SET TPU-DIP TO TRUE.
CALL "TPINITIALIZE" USING TPINFDEF-REC
USER-DATA-REC
TPSTATUS-REC.
IF NOT TPOK
error processing
⋮
make service call
receive the reply
check for unsolicited messages
⋮
CALL "TPTERM" USING TPSTATUS-REC.
IF NOT TPOK
error processing
⋮
EXIT PROGRAM.
```

上述例子中 TPINITIALIZE 用了 3 个参数。

（1）TPINFDEF-REC：COBOL 的 copybook 中定义的一组结构，传送用于注册的信息。

（2）USER-DATA-REC：用户数据，传送用于注册的用户信息。

（3）TPSTATUS-REC：也是 COBOLcopybook 中定义的一组数据结构，用于接收返回值。

如果成功，TPINITIALIZE 与 TPTERM 都将返回 TPOK，TPOK 这个开关变量定义在 TP-STATUS 中，是 COBOL copybook 中定义的 TPSTATUS-REC 的次级变量。

20.1.4 Tuxedo COBOL 服务器端编程

1. 服务器的初始化和终止

为了简化 ATMI 服务器端程序开发，Tuxedo 系统为一些模块预定义了控制程序，当执

行 buildserver-C 命令时，这个控制程序就被自动加载到服务器。这个控制程序是不能修改的。

在启动和退出这个控制程序时在服务器上做了以下事情。

（1）执行过程中忽略任何挂断（也就是忽略 SIGHUP 信号）。

（2）当收到标准操作系统中断信号（SIGTERM）时启动清理进程。

（3）附加到共享内存 BB。

（4）为进程创建消息队列。

（5）发布服务，这些服务可以是程序编译连接进来的，也可以是 Tuxedo 管理员在配置文件中指定的。

（6）处理命令行参数直到遇到代表系统参数结束的双破折号。

（7）调用 TPSVRINIT 来处理双破折号后面的参数，并且有选择地打开资源管理器。

（8）去服务请求队列中取请求消息，知道服务器被终止。

（9）当一个服务请求消息到达请求队列，主函数执行下面的任务。

① 如果有-r 参数，则记录服务请求的开始时间。

② 通知 BBL 此服务现在处于忙碌状态。

③ 调用特定的服务程序。

（10）当服务程序处理完它的输入时，执行下面的任务：

① 如果有-r 参数，则记录服务结束的时间。

② 更新服务被调用次数。

③ 去 BBL 中更新此服务的状态为准备被调用的状态。

④ 检查它的队列看有没有新的请求进来。

（11）如果服务需要停止，调用 TPSVRDONE 来执行终止服务操作。

注 意

对于 Tuxedo COBOL 服务器，因为是系统的控制程序来完成注册和注销的工作，所以不需要像在客户端程序中一样调用 TPINITIALIZE 或者 TPTERM ，否则将会出现错误，TP-STATUS 会被设置成 TPEPROTO 。

由于有了控制程序来处理系统相关的初始化和清理工作，如附加共享内存、创建消息队列等，使应用程序可以专注于业务逻辑。同时 Tuxedo 给应用程序提供了接口 TPSVRINIT，用于应用级别的服务器初始化，它的内容可以由用户编写，在服务器启动后，执行任何服务前，被 Tuxedo 自动调用；TPSVRDONE 用于应用级的服务器相关清理工作，它的内容可以由用户编写，在服务器收到终止命令，结束所有服务后，被 Tuxedo 自动执行。

用下面的方法来调用 TPSVRINIT。

示例 20-14：

```
LINKAGE SECTION.
01 CMD-LINE.
05 ARGC PIC 9(4) COMP-5.
05 ARGV.
```

```
10 ARGS PIC X OCCURS 0 TO 9999 DEPENDING ON ARGC.
01 TPSTATUS-REC.
COPY TPSTATUS.
PROCEDURE DIVISION USING CMD-LINE TPSTATUS-REC.
* User code
EXIT PROGRAM.
```

TPSVRINIT 常常用来建立 Tuxedo 服务器和资源管理器（RM），如数据库的长连接，直到服务器被终止，再断开连接，这样提高了数据库操作的效率。下面再举 TPSVRINIT 中打开 RM 的例子。

示例 20-15：

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TPSVRINIT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. USL-486.
OBJECT-COMPUTER. USL-486.
*
DATA DIVISION.
WORKING-STORAGE SECTION.
01 TPSTATUS-REC.
COPY TPSTATUS.
01 LOGMSG PIC X(50).
01 LOGMSG-LEN PIC S9(9) COMP-5.
*
LINKAGE SECTION.
01 CMD-LINE.
05 ARGC PIC 9(4) COMP-5.
05 ARGV.
10 ARGS PIC X OCCURS 0 TO 9999 DEPENDING ON ARGC.
01 SERVER-INIT-STATUS.
COPY TPSTATUS.
*
PROCEDURE DIVISION USING CMD-LINE SERVER-INIT-STATUS.
A-START.
. . . INSPECT the ARGV line and process arguments
IF arguments are invalid
MOVE "Invalid Arguments Passed" TO LOGMSG
PERFORM EXIT-NOW.
ELSE arguments are OK continue
CALL "TPOPEN" USING TPSTATUS-REC.
IF NOT TPOK
MOVE "TPOPEN Failed" TO LOGMSG
PERFORM EXIT-NOW.
```

```
SET TPOK IN SERVER-INIT-STATUS TO TRUE.
EXIT PROGRAM.

EXIT-NOW.
SET TPEINVAL IN SERVER-INIT-STATUS TO TRUE
MOVE 50 LOGMSG-LEN.
CALL "USERLOG" USING LOGMSG
LOGMSG-LEN
TPSTATUS-REC.
EXIT PROGRAM.
```

2．定义服务

在写服务程序时，必须要把 TPSVCSTART 的调用写在最前面，这个程序是用来接收和检查服务的参数和数据，下面是它的使用方法。

示例 20-16：

```
01 TPSVCDEF-REC.
COPY TPSVCDEF.
01 TPTYPE-REC.
COPY TPTYPE.
01 DATA-REC.
COPY User Data.
01 TPSTATUS-REC.
COPY TPSTATUS.
CALL "TPSVCSTART" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

服务的信息数据结构定义在 copybook TPSVCDEF 中，包含服务名称、通信句柄、客户端 ID 和请求数据等。

下面是一个典型的服务定义的例子。

示例 20-17：

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BUYSR.
AUTHOR. Tuxedo DEVELOPMENT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. USL-486.
OBJECT-COMPUTER. USL-486.
*
INPUT-OUTPUT SECTION.
⋮
*****************************************************
* Tuxedo definitions
*****************************************************
01 TPSVCRET-REC.
```

```
COPY TPSVCRET.
*
01 TPTYPE-REC.
COPY TPTYPE.
*
01 TPSTATUS-REC.
COPY TPSTATUS.
*
01 TPSVCDEF-REC.
COPY TPSVCDEF.
**************************************************
* Log message definitions
**************************************************
01 LOGMSG.
05 LOGMSG-TEXT PIC X(50).
*
01 LOGMSG-LEN PIC S9(9) COMP-5.
**************************************************
* User defined data records
**************************************************
01 CUST-REC.
COPY CUST.
*
LINKAGE SECTION.
*
PROCEDURE DIVISION.
*
START-BUYSR.
MOVE LENGTH OF LOGMSG TO LOGMSG-LEN.
OPEN files or DATABASE

**************************************************
* Get the data that was sent by the client
**************************************************
MOVE "Server Started" TO LOGMSG-TEXT.
PERFORM DO-USERLOG.
MOVE LENGTH OF CUST-REC TO LEN IN TPTYPE-REC.
CALL "TPSVCSTART" USING TPSVCDEF-REC
TPTYPE-REC
CUST-REC
TPSTATUS-REC.
IF TPTRUNCATE
MOVE "Input data exceeded CUST-REC length" TO LOGMSG-TEXT
PERFORM DO-USERLOG
PERFORM A-999-EXIT.
```

```
IF NOT TPOK
MOVE "TPSVCSTART Failed" TO LOGMSG-TEXT
PERFORM DO-USERLOG
PERFORM A-999-EXIT.
IF REC-TYPE NOT = "VIEW"
MOVE "REC-TYPE in not VIEW" TO LOGMSG-TEXT
PERFORM DO-USERLOG
PERFORM A-999-EXIT.
IF SUB-TYPE NOT = "cust"
MOVE "SUB-TYPE in not cust" TO LOGMSG-TEXT
PERFORM DO-USERLOG
PERFORM A-999-EXIT.
⋮
set consistency level of the transaction
⋮
*****************************************************
* Exit
*****************************************************
A-999-EXIT.
MOVE "Exiting" TO LOGMSG-TEXT.
PERFORM DO-USERLOG.
SET TPFAIL TO TRUE.
COPY TPRETURN REPLACING TPSVCRET-REC BY TPSVCRET-REC
TPTYPE-REC BY TPTYPE-REC
DATA-REC BY CUST-REC
TPSTATUS-REC BY TPSTATUS-REC.
*****************************************************
* Write to userlog
*****************************************************
DO-USERLOG.
CALL "USERLOG" USING LOGMSG
LOGMSG-LEN
TPSTATUS-REC.
```

示例 20-17 中，客户端发送的服务请求数据类型 REC-TYPE 是 VIEW， VIEW 的名字，即 SUB-TYPE 是 cust 。BUYSR 通过数据记录 CUST-REC 读取这些数据。

3．服务返回和转发

服务处理结束后，采用以下方式交出控制权。

（1）TPRETURN：给客户端一个返回。

（2）TPFORWAR：把当前请求转给另外的服务继续处理。

TPRETURN 和 TPFORWAR 调用了一个含有退出语句的 copybook，来表示本服务的结束。

TPRETURN 的内容如下。

示例 20-18：

```
01 TPSVCRET-REC.
COPY TPSVCRET.
01 TPTYPE-REC.
COPY TPTYPE.
01 DATA-REC.
COPY User Data.
01 TPSTATUS-REC.
COPY TPSTATUS.
COPY TPRETURN REPLACING TPSVCRET-REC BY TPSVCRET-REC
TPTYPE-REC BY TPTYPE-REC
DATA-REC BY DATA-REC
TPSTATUS-REC BY TPSTATUS-REC.
```

这里应该使用 COPY 而不是 CALL 来确保 EXIT 语句被调用，从而可以使 COBOL 程序把控制权返回给 Tuxedo 主程序。

TPSVCRET-REC 的结构如下。

示例 20-19：

```
05 TPRETURN-VAL            PIC S9(9) COMP-5.
88 TPSUCCESS VALUE 0.
88 TPFAIL VALUE 1.
    88 TPEXIT VALUE 2.
05 APPL-CODE               PIC S9(9) COMP-5.
```

TP-RETURN-VAL：TPSUCCESS 表示调用成功，把返回的数据放到调用者的数据记录中。TPFAIL 表示服务成功终止，返回调用者一个错误信息，设置 TP-STATUS 为 TPESVCFAIL；TPEXIT 表示服务终止不成功，发送错误给调用者，然后退出。

APPL-CODE：发送一个程序定义的返回码给调用者，调用者通过 TPSTATUS-REC 中的 APPL-RETURN-CODE 来取得 APPL-CODE，不论服务成功与否都会返回此信息。

如果客户端程序用 TPCALL 调用服务，并且服务成功执行了 TPRETURN，则消息会放到 O-DATA-REC，如果用的是 TPACALL 调用服务，并且服务 TPRETURN 成功返回，那么返回的消息存到 TPGETRPLY 的 DATA-REC 中。

如果在 TPRETURN 调用的过程中遇到了错误，服务就会把错误的信息发送回调用进程，调用者通过 TP-STATUS 来检查是否错误。

4．编译服务器程序

用 Tuxedo buildserver -C 生成一个 ATMI 服务器程序，编译时附带上所有相关的文件。具体 buildserver 格式如下。

示例 20-20：

```
buildserver -C -o filename -f filenames -l filenames -s svcname -v
```

其中-C 表示要编译的是 COBOL 程序，其他参数和编译 C 程序一样。

下面的编译是用 acct.o 应用程序文件产生一个服务程序 ACCT，它包含两个服务：NEW_ACCT（程序名为 OPEN_ACCT）和 CLOSE_ACCT（程序名跟服务名相同）。

示例 20-21：

```
buildserver -C -o ACCT -f acct.o -s NEW_ACCT:OPEN_ACCT -s CLOSE_ACCT
```

20.2　Tuxedo 下使用 COBOL 编程与 C 语言编程的异同

Tuxedo 的 COBOL 编程和 C 编程的思想是一样的，应用专注于业务逻辑，Tuxedo 负责处理系统连接、平台差异、网络通信等。

Tuxedo 的 COBOL 应用也和 C 应用一样都支持多种应用通信模式，包括请求/应答式通信（同步、异步、嵌套、转发）、会话通信、队列通信、事件代理通信和消息通知。

所不同的是 Tuxedo 的 COBOL 应用中传送数据使用的是类型记录（Typed Record）。而 C 应用中使用类型缓冲区（Typed Buffer），需要在 C 程序中分配和释放内存。虽然存在此差异，但是类型记录和类型缓冲区支持的类型都是一样的，包括 STRING、CARRAY、FML/FML32、VIEW/VIEW32 等。不过 Tuxedo 的 COBOL 程序不能直接操作 FML/FML32，因此如果对方程序需要用 FML/FML32 类型数据进行交互，在 COBOL 程序中就需要将 VIEW/VIEW32 类型记录转换成 FML/FML32 以后再发送。

在编译 Tuxedo 的 COBOL 客户程序或服务进程时，需要在 Tuxedo 的 build 命令行加上“-C”参数。编译后的程序的部署和 C 程序没有区别，即在 UBBCONFIG 配置文件中，不区分服务进程是 C 应用还是 COBOL 应用。需要注意的是，在 COBOL 程序的编译和运行时要设置 COBOL 编译器相关的环境变量。

还有极少的几个功能是 Tuxedo 的 C 应用支持，而 COBOL 应用不支持的。主要是 Tuxedo 支持 C 编程使用多线程，但对 COBOL 应用不支持多线程。

20.3　使用 COBOL 编写 Tuxedo 程序的局限性

20.3.1　FML 支持的局限性

因为 FML 是不支持 COBOL 语言的，需要 FML 依赖的 VIEW 来进行转换，才能供 C 语言使用。这就增加了 COBOL 语言编程的复杂性，导致了 COBOL 不能直接对 buffer 中的数据进行操作，但是 C 语言则可以直接操作 buffer 中的数据。

20.3.2　COBOL 语言编译的局限性

与在主机平台不同的是，在 Tuxedo 环境下的 COBOL 程序编译相对复杂，使用到的相

关文件需要单独编译，并且引用的程序和 copybook 不能通过库的方式来自动指定，需要编译时进行罗列。

20.3.3 开发人员要求比较高

只能通过 Tuxedo 提供的接口 ATMI 来跟其他程序交互，这需要在掌握业务的同时熟练地掌握 ATMI 对 COBOL 提供的 Functions，要求编程人员对 Tuxedo 有较高的认识，并且削弱了 COBOL 语言可读性强的特性。

20.3.4 错误处理开销

对错误及系统信息处理方面需要使用写日志的方式，不停地读写文件可能会加大系统资源开销。

20.3.5 数据类型的使用相对有限

因为考虑到底层语言是用 C 开发的问题，所以数据格式要经过转换才能被系统识别。在 COBOL 语言中经常使用的 COMP-3 类型的数据，需要注意与 C 的交互问题。

20.4 Tuxedo 下 COBOL 与 C 语言的混合编程及模块集成

一般来讲，可以在 COBOL 语言中调用 C 编写的函数，反过来也可以，下面介绍这两种语言之间的相互调用。

20.4.1 混合编程规则

1. 初始化环境

如果用 C 编写的程序，要调用 COBOL 程序，使用以下步骤。

（1）预先配置好 COBOL 环境在 C 编程环境中，这是必须的，因为其提供了最好的性能。

（2）把 COBOL 程序编为可执行的而不是 C 例行程序的一部分，每次如果想要调用一个 COBOL 程序，在 C 中按以下顺序编写。

① 加载程序。

② 调用程序。

③ 卸载程序。

2. 转换数据

一些 COBOL 数据类型和 C 程序的等效，但是其他的不一致。当在 COBOL 和 C 之间

传递数据时，确保约束交换合适的数据类型。

COBOL 中默认的参数传递是引用，如果是这样，C 将获得该参数的指针，如果传递是通过值传递，那么 COBOL 是传递实际的参数，但值传递只支持以下数据类型。

（1）字母数字 character。

（2）国家 character。

（3）BINARY。

（4）COMP。

（5）COMP-1。

（6）COMP-2。

（7）COMP-4。

（8）COMP-5。

（9）FUNCTION-POINTER。

（10）OBJECT REFERENCE。

（11）POINTER。

（12）PROCEDURE-POINTER。

20.4.2　COBOL 调用 C

以下例子是 COBOL 调用 C 函数。

（1）C 程序被调用是使用 COBOL 调用声明，这个声明不表明被调用的程序是用 C 或者还是 COBOL 编写的。

（2）COBOL 支持使用大小写混合的名字的程序调用。

（3）声明可以以不同的方式传递到 C 程序（如、引用调用、值调用）。

（4）必须定义函数返回值在调用声明中。

（5）传递的数据类型必须匹配。

示例如下。

示例 20-22：

```
 CBL PGMNAME(MIXED)
* This compiler option allows for
* case-sensitive names for called programs.
*
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "COBCALLC".
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
 01 N4                      PIC 9(4)  COMP-5.
 01 NS4                     PIC S9(4)  COMP-5.
 01 N9                      PIC 9(9)  COMP-5.
```

```
     01 NS9                     PIC S9(9) COMP-5.
     01 NS18                    USAGE COMP-2.
     01 D1                      USAGE COMP-2.
     01 D2                      USAGE COMP-2.
     01 R1.
        02 NR1                  PIC 9(8) COMP-5.
        02 NR2                  PIC 9(8) COMP-5.
        02 NR3                  PIC 9(8) COMP-5.
     PROCEDURE DIVISION.
        MOVE 123 TO N4
        MOVE -567 TO NS4
        MOVE 98765432 TO N9
        MOVE -13579456 TO NS9
        MOVE 222.22 TO NS18
        DISPLAY "Call MyFun with n4=" N4 " ns4=" NS4 " N9=" n9
        DISPLAY "             ns9=" NS9" ns18=" NS18
    *
    * The following CALL illustrates several ways to pass
    * arguments.
    *
        CALL "MyFun" USING N4 BY VALUE NS4 BY REFERENCE N9 NS9 NS18
        MOVE 1024 TO N4
    *
    * The following CALL returns the C function return value.
    *
        CALL "MyFunR" USING BY VALUE N4 RETURNING NS9
        DISPLAY "n4=" N4 " and ns9= n4 times n4= " NS9
        MOVE -357925680.25 TO D1
        CALL "MyFunD" USING BY VALUE D1 RETURNING D2
        DISPLAY "d1=" D1 " and d2= 2.0 times d2= " D2
        MOVE 11111 TO NR1
        MOVE 22222 TO NR2
        MOVE 33333 TO NR3
        CALL "MyFunV" USING R1
        STOP RUN.
```

C 程序如下所示。

示例 20-23：

```
#include <stdio.h>
extern void TPROG1(double *);
void
MyFun(short *ps1, short s2, long *k1, long *k2, double *m)
{
    double x;
    x = 2.0*(*m);
```

```
    printf("MyFun got s1=%d s2=%d k1=%d k2=%d x=%f\n",
            *ps1, s2, *k1,*k2, x);
}
long
MyFunR(short s1)
{
    return(s1 * s1);
}
double
MyFunD(double d1)
{
    double z;
    /* calling COBOL */
    z = 1122.3344;
    (void) TPROG1(&z);
    /* returning a value to COBOL */
    return(2.0 * d1);
}

void
MyFunV(long *pn)
{
    printf("MyFunV got %d %d %d\n", *pn, *(pn+1), *(pn+2));
}
```

20.4.3　C 调用 COBOL

以下程序是C调用COBOL的范例，文件tprog1.cbl是在C程序MyFun.c中函数MyFunD调用。

COBOL 源代码如下。

示例 20-24：

```
*
 IDENTIFICATION DIVISION.
 PROGRAM-ID. TPROG1.
*
 DATA DIVISION.
 LINKAGE SECTION.
*
 01 X              USAGE COMP-2.
*
 PROCEDURE DIVISION USING X.
     DISPLAY "TPROG1 got x= " X
     GOBACK.
```

编译并链接 COBOL 程序 cobcallc.cbl、tprog.cbl 和 MyFun.c，使用以下命令（环境：OS:AIX;COBOL：COBOL2）。

示例 20-25：

```
xlc -c MyFun.c
cob2 cobcallc MyFun.o tprog1.cbl -o cobcallc
```

运行 cobcallc，值如下。

示例 20-26：

```
call MyFun with n4=00123 ns4=-00567 n9=0098765432
             ns9=-0013579456 ns18=.22222000000000000E 03
MyFun got s1=123 s2=-567 k1=98765432 k2=-13579456 x=444.440000
n4=01024 and ns9= n4 times n4= 0001048576
TPROG1 got x=  .11223344000000000E+04
d1=-.35792568025000000E+09 and d2= 2.0 times d2= -.71585136050000000E+09
MyFunV got 11111 22222 33333
```

第 21 章　基于 Tuxedo 对大机应用的迁移——ART

21.1　ART 简介

ART 是 Application Runtime 的简称，是一个解决 IBM 大型机应用迁移的专业工具。将大型机应用程序迁移到开放系统避免了应用程序重写的成本和风险，降低大型机 MIPS 消耗，保持大型机级别的服务质量，借助灵活的、支持 SOA 的环境以提高业务敏捷度。

ART 组件包括以下内容。

（1）Oracle Tuxedo Application Runtime for CICS and Batch。

① CICS API。

② 终端仿真和服务。

③ 批处理作业执行环境。

（2）Oracle Tuxedo Application Rehosting Workbench。

① 应用程序信息库与编目器。

② 语言迁移工具：COBOL、JCL。

③ 数据迁移工具：VSAM 和普通文件、DB2。

21.2　Application Rehosting Workbench 作业运行环境

21.2.1　关键特性

（1）自动化：提高生产力。

（2）精确性：降低错误率。

（3）高效：自动转换数千万行代码。

（4）可重复性：确保多次重复的结果一致。

（5）可扩展性：方便修改和添加转换规则。

21.2.2　优点

（1）一致性、高效率。

（2）将项目成本和风险降至最低。
（3）处理大规模应用资产。
（4）统一的转换，易于维护。
（5）支持用户特定的转换，易于扩展。

21.2.3 流程简介

ART 流程简介如图 21-1 所示。

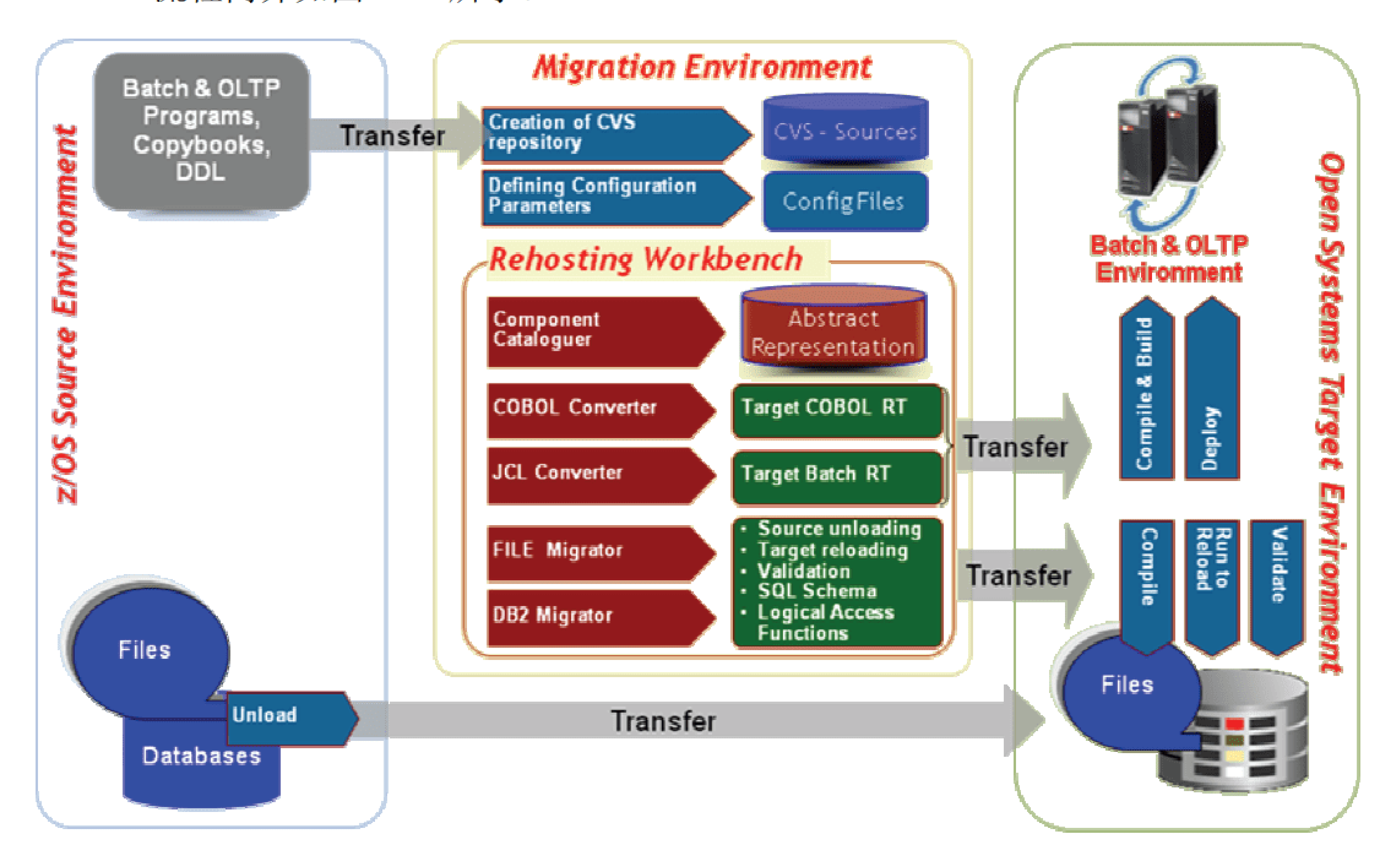


图 21-1

21.2.4 详细流程

1．应用编目和分析（Cataloger）

（1）功能

① 分析所有资源组件。

- 探查内在的不一致。
- 为组件创建内部表示方法。

② 分析组件之间的引用关系。

- 探查资源级别的不一致。
- 分析组件之间的交叉引用。

③ 报告资源清单以及不一致之处。

- 包括未使用或者丢失的组件。

（2）配置

① 系统描述文件。

- 描述了迁移平台所有的源码文件的位置和类型。
- 描述引用资源的位置。
- 描述各种转换以及分析的选项。

② Hint 参数文件。

- 补充的一些引用信息。
- 可以克服静态分析的一些局限性而获得资源的真实结构图。

（3）命令

① Catalog。

- 和 preparse、analyze、fast-final 相结合的命令。

② preparse/preparse-files。

- 转换资源生成内在的.pob 描述文件（Persistent Object Base）。
- 可并行运行。

③ analyze。

- 分析.pob 文件。
- 构建 Cataloger 描述表（生成 symtab-<ProjectName>.pob 文件）。

④ fast-final。

- 运行 post-analysis 以及生成报告。
- 分析组件之间相互引用关系。
- 指出引用关系之间的不一致。
- 生成报告。

2. DB2 到 Oracle 的数据迁移

（1）功能

转换 z/OS DB2 对象到开放平台 Oracle 对象

① 迁移所有的 SQL 对象。

② 支持所有常用数据类型。

③ 处理保留字（重命名）。

④ 相关处理过程如下。

- 使用 DSNTIAUL 下载 DB2 表。
- 用逻辑名自动处理 8 字符长名字限制。
- 转码。
- 导入数据。

（2）配置

① 系统描述文件。

② 数据库参数文件。

- 列出要重名的表或者列。

3. 文件的迁移

（1）功能

转移 z/OS 文件（VSAM、QSAM、GDG）到目标平台的 ISAM 文件或者 ORACLE 数据库。

① 下载物理文件到目标平台。

② 转码（EBCDIC-to-ASCII）以及导入数据。

③ 执行初步检查。

④ 将文件转换到相关的 DBMS。

- 翻译成关系模型。
- 优化关系模型。
- 转换应用代码来连接关系型数据库。

（2）配置

① 系统描述文件。

② 数据库参数文件。

③ 其他配置。

- 对于每一个要迁移的文件，都要在 COBOLcopybook 中描述。
- Datamap-<configuration name>.re 列出所有物理文件名，以及每个文件的类型。
- mapper-<configuration name>.re 列出所有文件，指定相应 COBOL 描述以及区别的规则。

4. COBOL 程序迁移

（1）功能

转换 IBM COBOL 源程序到开放平台的 COBOL 程序。

① 转换 IBM Enterprise COBOL 程序为 MicroFocus COBOL 程序（或其他开放平台 COBOL 程序）。

② file-to-Oracle 转换匹配，组件重命名。

③ 嵌入式 SQL 转换。

④ 把程序中的 EXEC CICS 命令正规化。

（2）配置

① 系统配置文件。

② 主配置文件。

- 各种参数，如是否在转换后合并 copybook。

③ 从属配置文件，可能是手写的也可能是工具生成的。

- 组件重命名。
- 重命名标识符。
- File-to-Oracle 转换信息。

5. JCL 脚本转换

（1）功能

① 转换 IBM JCL jobs（主文件、PROCs、INCLUDEs、SYSIN 文件等）到目标平台的脚本。

- Ksh 脚本语言。
- ART 脚本运行环境的组件调用（程序执行、文件分配、文件处理、应用调用等）。

② 生成相关源程序。

- **JCL**　主 JCL 文件，这些文件定义了一个或多个 JCL job。
- **JCL-Lib**　JCL 引用文件，定义 EXEC 调用的过程，或者 INCLUDE 的引用。
- **JCL-Sysin**　应用程序运用的 SYSIN 文件。

（2）配置

① 系统配置文件。

② 主配置文件。

- 各种参数，如数据文件根目录、排序应用。

③ 从属文件，可能是手写的也可能是工具生成的。

- 组件重命名。
- 重命名标识符。
- File-to-Oracle 转换信息。

21.3　ART for CICS 作业运行环境

21.3.1　关键特性

（1）基于成熟的 Tuxedo 产品。

（2）支持多节点、多平台、多种网格协议。

（3）线性可伸缩。

（4）开放、SOA、可扩展。

（5）集中管理。

21.3.2　优点

（1）具备大型机的性能（RASP），保持服务质量。

（2）只需相当于大型机成本的一小部分即可。

（3）灵活，能够满足未来需求的基础架构。

（4）更快速、经济的集成，更高的重用率和开发人员效率。

21.3.3　流程简介

应用角度流程如图 21-2 所示。

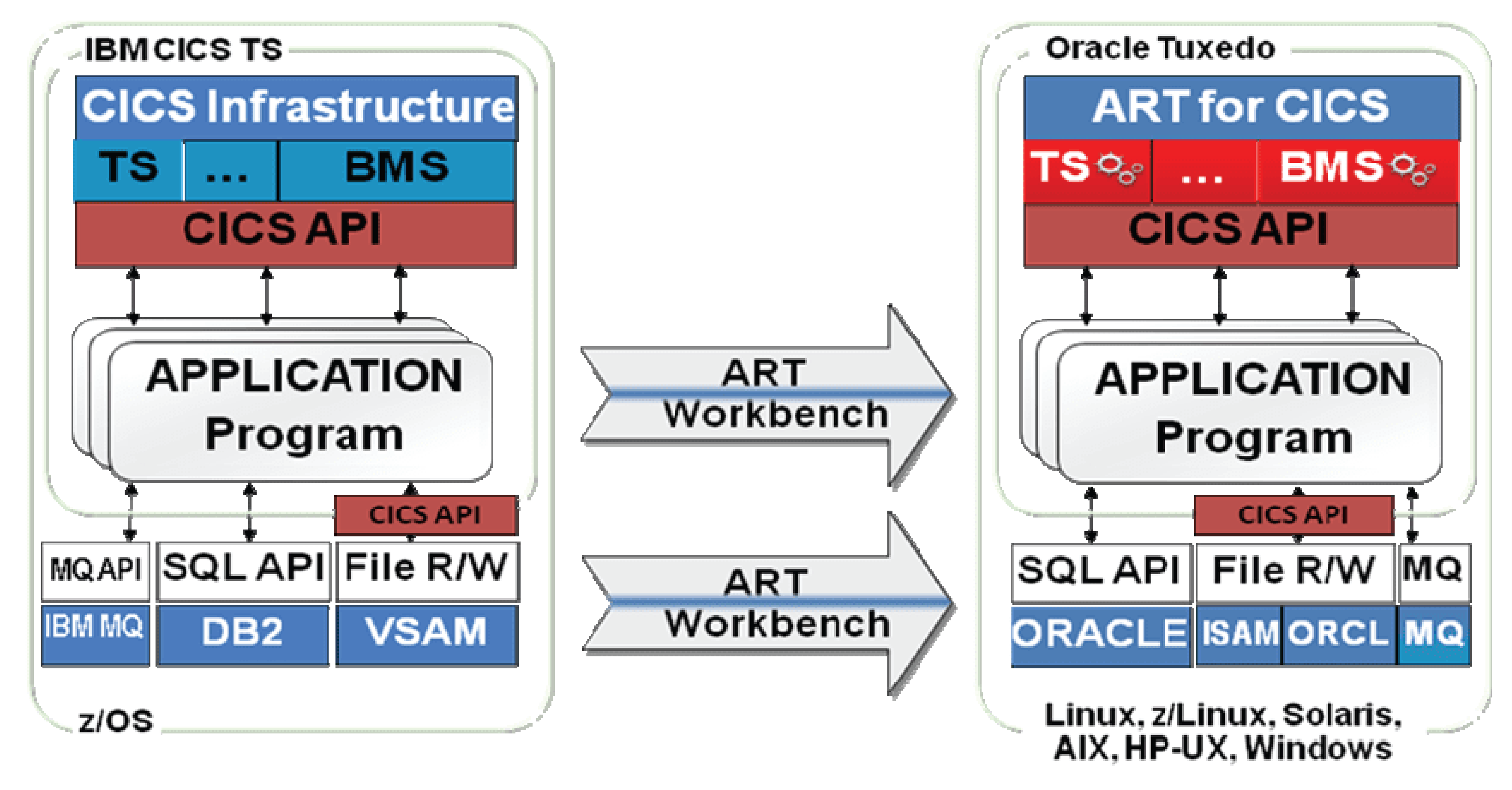


图 21-2

系统角度流程如图 21-3 所示。

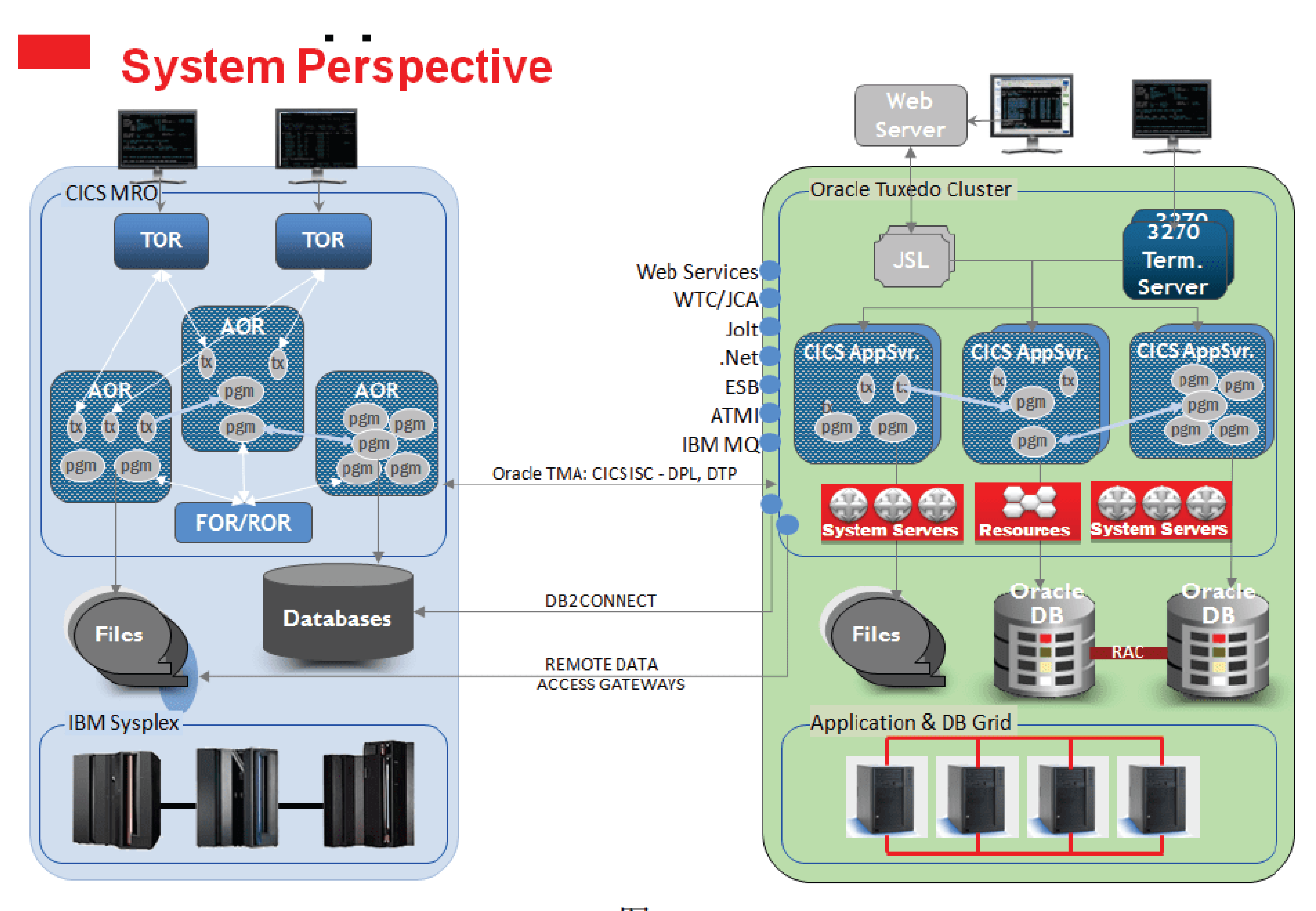


图 21-3

21.3.4　详细流程

1. ART CICS Runtime 简介

（1）架构

① CICS 是一个交易中间件。

② 当使用 ART workbench 从 Z/OS 移植到 UNIX/Linux 平台时，ART CICS Runtime 提供了 CICS 运行环境和 3270 终端仿真。

（2）ART CICS 运行环境主要组成部分

① CICS 预处理和运行库。

② CICS Runtime Tuxedo 服务以及资源配置文件。

（3）ART CICS 运行库

① 在 Z/OS CICS 应用中，所有的操作和资源管理都是通过 EXEC CICS API 来实现的。

② ART CICS 提供了执行 EXEC CICS API 的运行库。

（4）ART CICS Tuxedo 服务进程

ART CICS 基于 Tuxedo 的服务进程和服务来运行迁移后的 CICS 交易（Transaction），管理 3270 仿真终端的连接等。

① 必要的服务进程。

- **终端连接进程（ARTTCPH 和 ARTTCPL）** 管理 3270 仿真终端接入 ART CICS 应用。
- **连接进程 ARTCNX** 管理用户的会话以及提供系统 Transaction(CSGM: Good Morning Screen, CESN: Sign On, CESF: Sign off）。
- **同步交易服务进程 ARTSTRN** 管理标准的 CICS 同步交易。

② 可选性服务进程。

- **同步交易服务进程 ARTSTR1** 管理不可以同时运行，但可以顺序执行的 CICS 同步交易。
- **异步交易服务进程 ARTATRN 和 ARTATR1** 和 ARTSTRN 、ARTSTR1 相似，但是服务于异步的交易，即以 EXEC CICS START TRANSID 开始的交易。
- **TS Queue 服务进程 ARTTSQ** 管理 CICS TS queue-通过 CICS 专门的命令管理的文件。
- **TD Queue 服务进程 ARTTDQ** 管理 CICS TD queue。

③ 服务进程的配置。

- 这些服务进程都是 Tuxedo SERVER，配置在 UBB 中。
- ART CICS 服务进程会读取配置在 CICS Runtime 资源配置文件中的信息。

④ ART CICS Runtime 资源管理文件。

- z/Os CICS 资源管理。

在 Z/OS 中，所有 CICS 应用使用的组件（terminals、transactions、programs、maps、files 等）都要用一个名为 CSD 的声明文件来声明。

每个资源都必须属于一个资源组，这样就能让一组资源连接在一起来组成一个功能或应用来管理（install、delete、copy 到另外的 CSD...）。

- ART CICS Runtime 资源管理。

ART CICS Runtime 只管理在 Z/OS 的 CICS CSD 文件中预先定义的资源信息的子集，这些文件都存放在 UNIX 的指定目录中。

CICS 管理以下资源。

① 交易类 Tranclasses（transclasses.desc 文件）：该文件包括 CICS 交易相关的所有类，在 ART CICS Runtime 中，tranclass 提供一种定义是否几个相同交易的实例能够并行执行的机制。

② 交易 Transaction（transaction.desc 文件）：一个交易属于一个 transaction class，每个交易要在这个文件中指定其对应的程序名。

③ 程序 Program（program.desc 文件）：该文件包含交易对应的程序，或通过 LINK 或者 XCTL 调用的程序。

④ TS Queue 模型（tsqmodel.desc 文件）：包含所有 CICS 程序中用到的 TS queue 所关联的 TS queue 模型。这些模型用作定义 TS queue 是否可恢复，以及存储在数据库、文件，还是内存。

⑤ Mapsets（mapsets.desc 文件）：该文件中包含了 CICS 应用所关联的所有的 Mapset。Mapset 是包含一个或多个 screen（用于终端和 CICS 程序之间的交互）的物理组件。这些资源被专门的 CICS 语句如 EXEC CICS SEND 或者 RECEIVE MAP 使用。

⑥ Typeterms（typeterms.desc 文件）：包含所有 3270 终端的类型，CICS Runtime TCP 进程支持这些类型。

2. ART CICS Runtime 配置

（1）ART CICS Runtime 配置

① 可以使用 UNIX 系统中的~/.profile 文件，定义 CICS Runtime 使用的或者 Tuxedo 使用的 PATH 以及其他环境变量。

② 也可以在 Tuxedo 的 envfile 中为某些服务进程定义特定的环境变量。

③ 在 Tuxedo 的 UBBCONFIG 中配置所需的 CICS Runtime 服务进程。

④ 配置 ART CICS Runtime 使用的 CICS 资源。

（2）ART CICS 使用的环境变量

ART CICS 使用的环境变量见表 21-1。

表 21-1

变量名	取值	用途	环境
TUXDIR	在安装时设置	必需。指定 tuxedo 安装目录，默认 /usr/tuxedo	TUXEDO
TUXCONFIG	在安装时设置	必需。tuxconfig 文件的全路径	TUXEDO
KIXDIR	在安装时设置	必需。CICS Runtime 组件的绝对路径	CICS Runtime
APPDIR	${KIXDIR}/bin	必需。 CICS Runtime Servers 二进制文件的路径	CICS Runtime
KIXCONFIG	在安装时设置	必需。CICS Runtime 资源配置文件的路径	CICS Runtime
KIX_TS_DIR	在安装时设置	必需。不可恢复的 CICS Queue TS 目录	CICS Runtime

（3）Tuxedo UBBCONFIG 文件

必需的 ART CICS 服务进程包括 ARTTCPL 和 ARTCNX。

示例 21-1：

```
*SERVERS
ARTTCPL SRVGRP=TCP00 SRVID=101 CLOPT="-o /home2/work9/demo/Logs/TUX/
sysout/stdout tcp -e /home2/work9/demo/Logs/TUX/sysout/stderr tcp -- -M 4
-m 1 -L //deimos:2994 -n //deimos:2992"
ARTCNX SRVGRP=GRP01 SRVID=15 CONV=Y MIN=1 MAX=1 RQADDR=QCNX015 REPLYQ=Y
CLOPT="-o /home2/work9/demo/Logs/TUX/sysout/stdout_cnx -e /home2/work9/
demo/Logs/TUX/sysout/stderr_cnx -r --"
```

（4）ART CICS Runtime 资源配置文件

必需的 ART CICS 资源配置文件包括 typeterms.desc 和 mapsets.desc。

① typeterms.desc 配置文件。

该文件用于 ARTTCPL，定义了不同种类的终端属性。

示例 21-2：

```
[typeterm]
name=IBM-3279-5E
color=YES
defscreencolumn=80
defscreenrow=24
description="IBM 327x family terminal"
hilight=YES
logonmsg=YES
outline=NO
swastatus=ENABLED
uctran=NO
userarealen=0
```

② mapsets.desc 配置文件。

该文件的以下配置用来启动 CSGM 交易画面，

示例 21-3：

```
[mapset]
name=ABANNER
filename=<KIXDIR>/sysmap/abanner.mpdef
```

21.4　ART for Batch 作业运行环境

21.4.1　流程简介

应用角度流程如图 21-4 所示。

系统角度流程如图 21-5 所示。

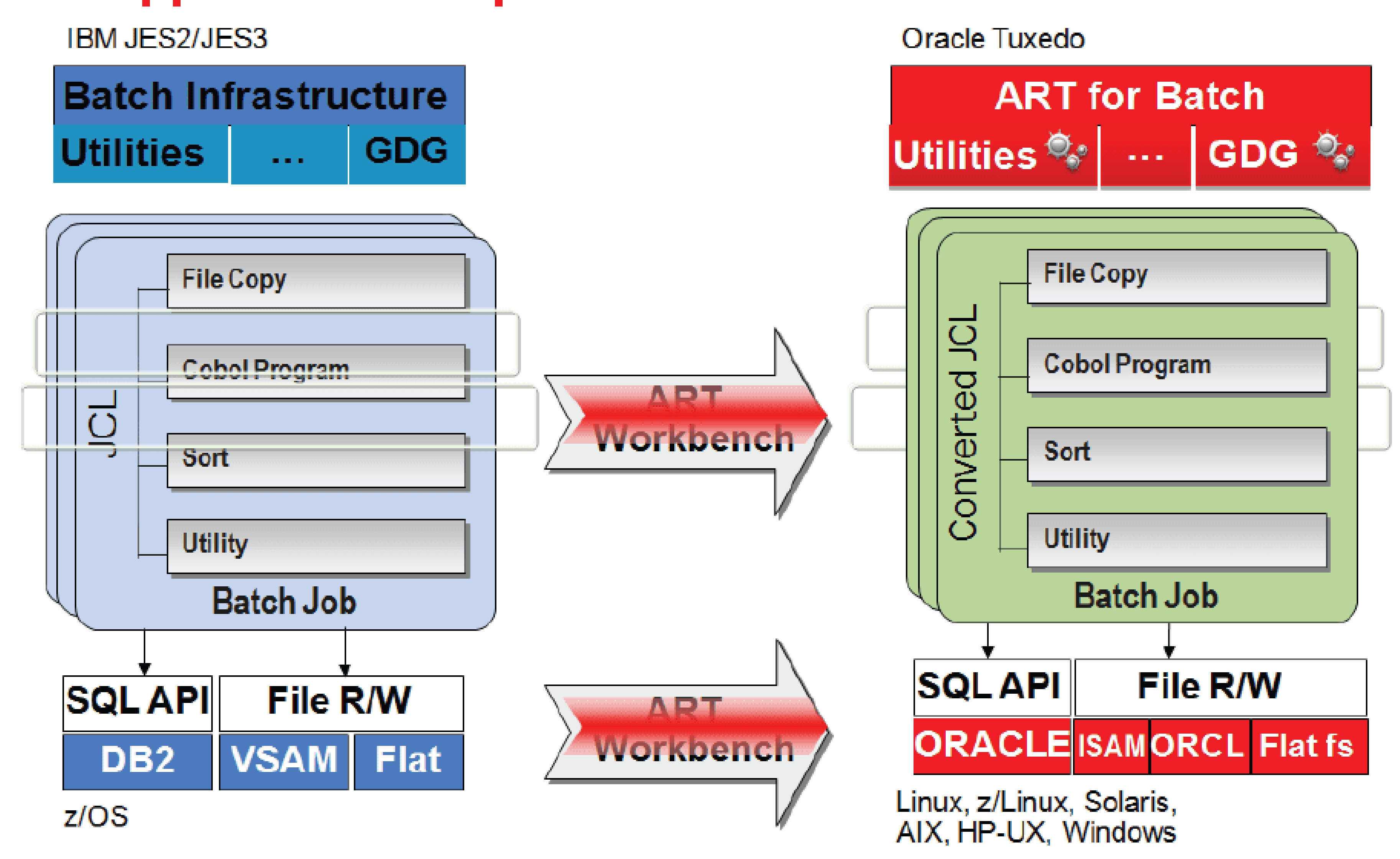


图 21-4

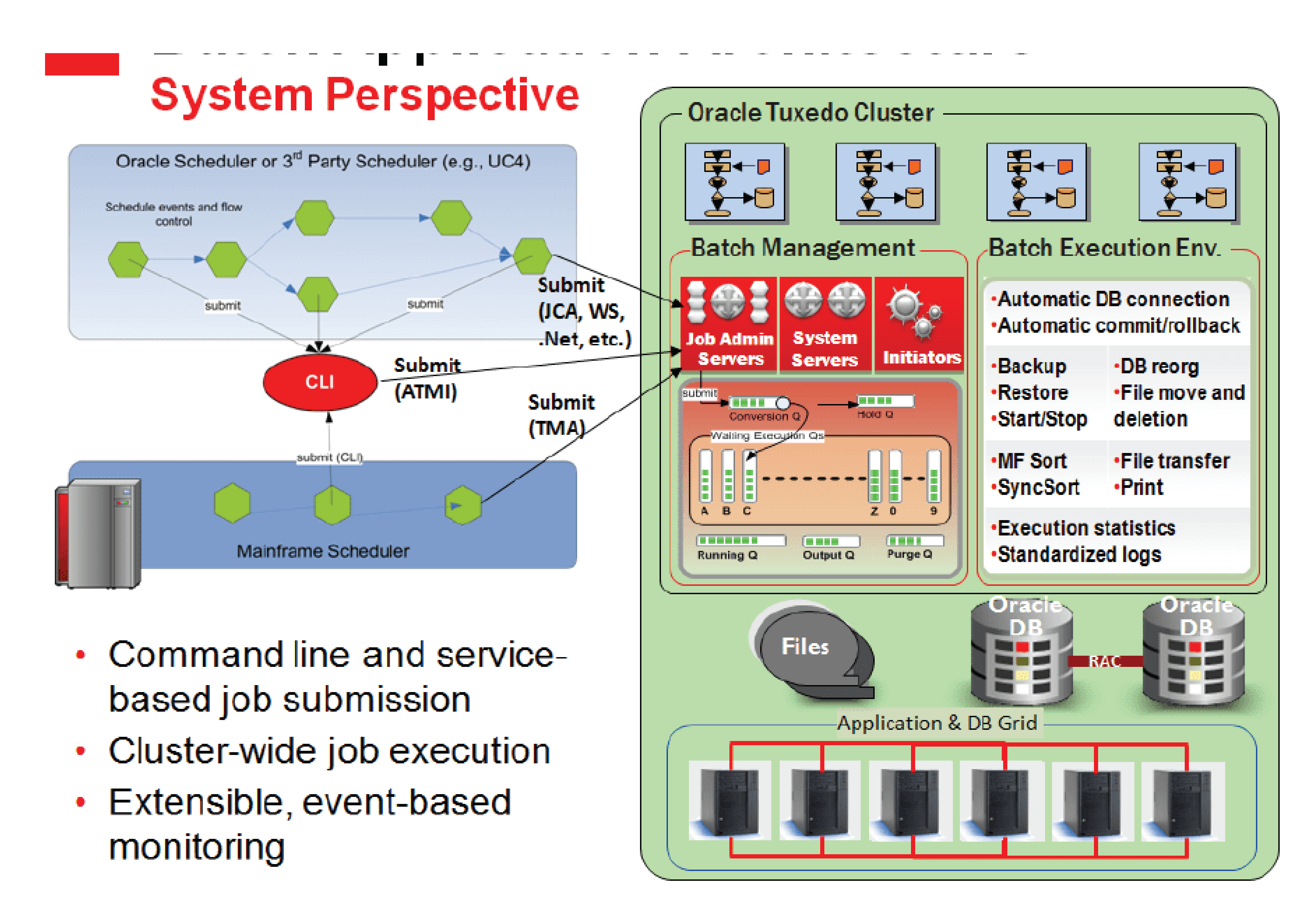


图 21-5

21.4.2 详细流程

1．概述

ART Workbench 将 z/OS JCL 转换成可以在开放平台上运行的 KSH。当在 ART Batch 运行环境中提交一个作业时，这个 KSH 脚本通过以下步骤执行。

（1）输入。在这个步骤中，要分析作业的参数。

（2）转换。这个过程中，执行以下动作：① 加入所有的外部 KSH（procedures 和/或 includes）；② 用当前值代替脚本中用到的变量。

（3）执行。在运行环境中执行该脚本。

2．运行环境

ART for Batch 环境变量见表 21-2。

表 21-2

变量	用途
DATA	指出数据存放地方
TMP	应用文件的暂存目录
SPOOL	输出目录
PROCLIB	PROC 和 INCLUDE 文件目录
MT_ACC_FILEPATH	文件之间并发访问
MT_DB_LOGIN	数据库连接用户
MT_LOG	日志目录
MT_TMP	内部文件的暂存目录

3．创建一个脚本

（1）脚本的结构

ART for Batch 提供了一个脚本模板，这个模板中指出了每个过程的具体步骤，用以规范使用 KSH。KSH 脚本中每一个步骤（STEP）都对应一个标号（LABEL），一个步骤结束后，JUMP_LABEL 变量的值改变，以执行下一个步骤。

（2）定义和使用 symbol

symbol 是一个脚本文件中的变量，ART Batch 的 KSH 中，变量可以使用$[symbol]来定义，可以通过 m_SymbolSet 来给变量赋值。

（3）创建执行程序步骤

一个步骤（STEP），是调用一系列的运行环境的功能来完成一个具体的动作。最常见的就是执行一个应用程序。一个作业中通常包含多个步骤。

4．创建一个过程

（1）过程简介

ART for Batch 中的过程和 Z/OS 中的过程（Procedure）原则上一致。

（2）过程优点

① 一次编写，然后可以多次调用。

② 过程可以统一修改。

（3）过程分类

① In-stream 过程：包含在脚本之中，但这个过程只能用在当前的脚本中。

② External 过程：写在其他的文件中，与脚本文件分开，这样可以被多个脚本使用。

（4）创建一个 In-stream 过程

该过程定义在文件最后，也就是到 m_JobEnd 后。在 KSH 脚本中以 m_ProcBegin 开始，期间进行一些具体操作后，以 m_ProcEnd 结束。

（5）创建一个 External 过程

该过程就不使用 m_ProcBegin 和 m_ProcEnd，在过程中就是简单的代码，为了显示这个是过程代码，可以开始于如下代码。

示例 21-4：

```
JUMP LABEL=FIRSTSTEP
 ;;
(FIRSTSTEP)
```

结束于如下代码。

示例 21-5：

```
JUMP_LABEL=ENDPROC
;;
(ENDPROC)
```

① 使用过程。

在一个 KSH 脚本中使用一个过程，要用 m_ProcInclude 来调用，在 Batch 执行的转换阶段，所调用的过程会被展开到主脚本中。

② 在执行时修改过程。

过程中的任务在执行时可以根据运行状态改变，通过修改 symbol 或者指定 parameter 这两种方法都可以实现。

5．控制脚本的执行

（1）控制执行的步骤

① 使用 m_CondIf、m_CondElse 和 m_CondEndif 可以控制一个或多个步骤的执行，相当于 Z/OS 中的 IF、THEN、ELSE 和 ENDIF。

② 使用 m_CondExec 可以控制是否执行该步骤，m_CondExec 必须有一个或多个参数，执行的条件是该参数或者所有参数都符合时才执行。第一，m_CondExec 4，LT，STEPEC01 表明是在返回结果符合要求才执行；第二，m_CondExec EVEN 表明就算前一步非正常中断了也执行接下来的工作；第三，m_CondExec ONLY 表明只有在前一步非正常结束时才执行接下来的工作。

（2）改变默认的输出信息

如果 Batch 管理员想改变默认的输出信息，可以通过 MT_DISPLAY_MESSAGE_FILE 指定的配置文件来修改。这个文件是用分号隔开字段的 CSV（comma separated values）文件，在文件中每个记录都分成 6 个部分：① 信息定义；② 能显示该信息的函数（默认“*”）；③ 显示级别；④ 显示的目的；⑤ 保留以后用；⑥ 显示的信息。

6．文件使用

（1）创建文件定义

使用 m_FileBuild 函数定义一个文件，支持 3 种文件：顺序文件、顺序行文件、索引文件。

（2）分派以及使用文件

在运行中，一个文件可以被一个函数使用也可以被一个程序使用，在使用前，一定要用 m_FileAssign 指定：指定打开的模式，指定访问模式，指定该文件是否为生成文件，指定访问的逻辑名所映射到的具体物理位置。

（3）使用 GDG 文件

① 使用 m_GenDefine 函数来指定，其中唯一的参数-s 用来指定该文件的最高版本。

② 可以使用 m_FileAssign 函数的-g 参数指明该文件是 GDG 文件。

（4）使用 In-Stream 文件

文件的数据直接从 KSH 脚本中写入，使用-i 参数，以_end 结束 in-stream 流。

（5）使用连接文件集

m_FileAssign 函数-C 参数指定使用一系列文件作为连续输入（在 Z/OS 的 JCL 中作为 DD card，该 DD card 只有第一个含有 label）。

（6）使用外部的“sysin”

使用 m_UtilityExec 函数可以使用含有执行命令的“external sysin”文件。

（7）删除文件

使用 m_FileDelete 可以删除文件。

（8）RDB 文件

z/OS 文件可以通过 ART Workbench 转换为关系型数据库表，.rdb 文件描述了转换的对应关系。

（9）使用 RDBMS 连接

① 当执行程序要使用到数据库连接时，可以使用 m_ProgramExec 的-b 参数。

② 数据库的创建连接、断开连接、提交、回滚都是隐式执行的，数据库连接串通过环境变量 MT_DB_LOGIN 指定，用户名和密码必须正确。

7．用 EJR 提交一个 JOB

TuxJES 可以发起一个 JOB，一个 JOB 也可以直接通过 EJR 发起。

在执行前，确保整个环境的正确性，包括环境变量和目录。

后　　记

作为企业级 IT 运维系列中的《Tuxedo 企业级运维实战》如果能抛砖引玉，带给读者思路和启迪，则足以欣慰。就像我们回溯历史经历和切身之事时，经常感觉自身的渺小，就如同地球之比于宇宙，小得像太平洋上的一滴水珠。

而这个世界上，最接近永远不变的，恰恰是“变化”本身。随着产品的更新，新技术的出现，以及厂商之间的商业整合，很多具体的技术细节都会随之发生变化，届时也会跟随有相应的修订，但其中涉及的基本概念和技术思路却是相通的。道为本，术为末，始能根深叶茂。

更多实战经验书籍，请参看叱咤风云同系列的《WebLogic 企业级运维实战》和《GoldenGate 企业级运维实战》等。